PRACE

PARTNERSHIP FOR
ADVANCED COMPUTING
IN EUROPE

2021

summerofhpc.prace-ri.eu

A long hot summer is time for a break, right? Not necessarily!
PRACE Summer of HPC 2021 reports by participants are here.

Remote 2021!

Leon Kos

Summer of HPC 2021 was the largest edition so far. Started all virtual with 65 participants and their mentors at 14 PRACE HPC sites working on 33 projects.

Summer of HPC is a PRACE programme that offers summer placements at HPC centres across Europe to late-stage undergraduate and master's students. Total of 65 top applicants from across Europe were selected to participate in pairs on 33 projects supported and mentored online from 14 PRACE hosting sites. Participants spent two months working on projects related to PRACE technical or industrial work and produce a report and video of their results. A kick-off online training week (see photo) was organised by Irish Centre for High End Computing (ICHEC).

PRACE SoHPC2021 Coordinator
Leon Kos, University of Ljubljana
Phone: +386 4771 436 E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCMore Information
http://summerofhpc.prace-ri.eu

Leon Kos

Contents

Analysis of memory scheduling policies

Aneta Ivaničová

The ACME accelerator architecture.

Main memories have become performance bottlenecks, which means that processors are spending their time waiting for memory operations. For this reason, we are presenting an analysis of the memory scheduler called BLISS in this article.

The memory bandwidth is the maximum amount of data that can be received and written to the memory or read from the memory and returned to the processor per unit of time.[3] The memory throughput, the actual transfer rate, which is limited by the memory bandwidth, has made main memory a performance bottleneck. Although maximizing memory throughput is beneficial in some cases, in other cases it might degrade overall system performance. In a system where multiple cores share a common memory interface, concurrent memory requests from different threads executing on different cores can interfere with each other while accessing the shared DRAM main memory system. This is also called inter-thread interference and it degrades system performance and slows down applications. To mitigate inter-thread interference we decided to study four state-of-the-art memory schedulers, then we picked one of them, thoroughly analyzed it, implemented it in an open source RISC-V ISA based simulator named Coyote, then tests began, and after that evaluation was expected to start. Our choice for the memory scheduler was the Blacklisting Memory Scheduler called BLISS. Among the reasons were that BLISS helps to achieve higher system performance and fairness, while incurring low hardware cost and scheduling latency. Also, BLISS does not implement total order-based ranking of memory requests, which leads to lower complexity, so that strict double data rate memory timing protocols can be met.

MEEP, ACME and Coyote

The project revolves around MEEP (MareNostrum Experimental Exascale Platform) which is a flexible FPGA-based emulation platform that serves as a basis for creating European-based chips and an infrastructure to enable rapid prototyping. The MEEP project is currently emulating a self-hosted accelerator called ACME (Accelerated Compute Memory Engine), which has two main components: the VAS tiles and the memory tiles. The VAS tile is a cluster of 8 scalar cores, each core supports a Vector Processor Unit (VPU) and two Systolic Array units. A scalar core has its own L1 instruction and data caches, and a L2 data cache slice which also can act as a scratchpad. A memory tile consists of a memory controller, a slice of high bandwidth memory, a JTLB (address-translation cache) and the MCPU which is designed to manage memory requests and related operations. ACME aims to improve the performance of dense (compute-bound) and sparse (memory-bandwidth-bound) workloads, and to find the balance between the memory hierarchy design and the number of fused multiply-add (FMA) units available in the system that the performance depends on. MEEP proposes Coyote which is a performance modeling tool based on two open source simulators called Sparta and Spike. Coyote provides detailed insights at various levels and granularity, while focusing on data movement and the modeling of the memory hierarchy of the system.[5]

The main memory

In case of ACME, the main memory is a high bandwidth memory (HBM), which is a stacked double data rate (DDR) memory that is connected via an interposer to an FPGA. At the bottom of the

HBM is a base logic die. On the top of the logic die are stacked DRAM dies, which are connected through-silicon vias (TSVs), as shown in Figure 1. Each slice of core DRAM die has two channels with eight independent bank groups. The utilization of memory banks enables concurrent DRAM main memory accesses and increases memory bandwidth. All banks within a channel share the command, address and data buses of the channel. Each bank has a structure of a two-dimensional array of rows and columns.

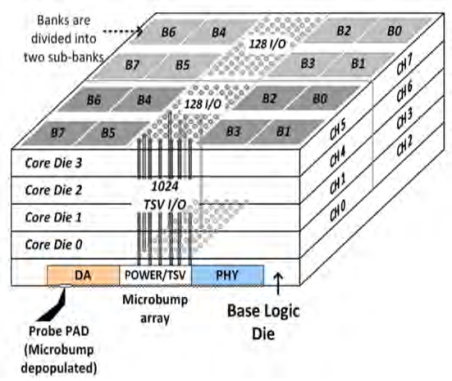


Figure 1: The high bandwidth memory architecture.[4]

Memory access terminologies

Before moving on to the memory scheduler we need to explain a term called a row-buffer hit. So, when a processor generates a memory request, the data is searched after in all levels of cache memory and if the data is not found in the last level cache (LLC), then the memory controller looks for the data in the main memory by sending the physical address of the data via the memory address bus. On a data access, the entire row containing the data is copied into a structure called the row buffer. A subsequent access to the same row can be served from the row buffer itself and it does not need to access the array. This is called a row-buffer hit. On the other hand, when accessing a different row, the previous row of data must be returned, and then the next row can be accessed. This type of access is called a row-buffer miss or conflict.

The memory scheduler

To mitigate inter-thread interference, BLISS separates threads into two groups: interference-causing and vulnerable-to-interference. When a large number of consecutive requests are served from the same thread other threads will likely stall, therefore BLISS counts the number of consecutive requests served from the same thread. When this count exceeds a threshold, BLISS places the thread into the interference-causing group, also called the blacklisted group, and other threads are placed in the vulnerable-to-interference group. BLISS has two components: The Blacklisting Mechanism and The Memory Scheduling Mechanism. The Blacklisting Mechanism needs to keep track of the following three quantities: the Thread ID of the last scheduled request, the Number Of Requests Served from a thread, and the Blacklist Status of each thread. Before a request is issued by the memory controller, it compares the thread ID of the current request and the Thread ID of the last scheduled request. If they are the same, then the Number Of Requests Served counter is incremented. However, if they are different then the counter is reset to zero and the Thread ID register of the last scheduled request is updated with the thread ID of the current request. If the Number Of Requests Served counter exceeds a blacklisting threshold, which can be four according to the research papers, then the thread ID of the current request is blacklisted, and the counter is reset to zero. The blacklisting information is cleared periodically after every Clearing Interval, which is ten thousand cycles in the papers. In addition, this information is used by the Memory Scheduling Mechanism to determine the scheduling priority of a request. Requests from threads that are placed in the vulnerable-to-interference group or also called non-blacklisted threads are prioritized, then row-buffer hit requests follow because they optimize bandwidth utilization, and finally older requests are prioritized for forward progress. It is important to prioritize threads which are vulnerable to interference because they are compute-bound, therefore it is beneficial when they spend less time waiting for memory operations.

Conclusion

BLISS optimizes applications by exploiting the imbalance in number of memory requests among the threads. Threads that have less number of memory requests are prioritized over threads which in comparison are more memory-intensive. Since applications ported to Coyote (AXPY, Matmul, Somier, SPMV) exhibit little imbalance in terms of number of memory requests among its threads, it would be interesting to port a few applications that could demonstrate suitable use cases for the BLISS optimization. We hope to do this in our future work for evaluation purposes.

References

[1] Fell, A., Mazure, D., Garcia, T., Pérez, B., Teruel, X., Wilson, P., & Davis, J. (2021). The MareNostrum Experimental Exascale Platform (MEEP). *Supercomputing Frontiers And Innovations*, 8(1), 62-81. doi:http://dx.doi.org/10.14529/jsfi210105

[2] Subramanian, L., Lee, D., Seshadri, V., Rastogi, H., & Mutlu, O. (2014, October). The blacklisting memory scheduler: Achieving high performance and fairness at low cost. In 2014 IEEE 32nd International Conference on Computer Design (ICCD) (pp. 8-15). IEEE.

[3] John Burke 2015, TechTarget, viewed 28 August 2021, <https://searchnetworking.techtarget.com/definition /throughput>

[4] Jun, H., Cho, J., Lee, K., Son, H. Y., Kim, K., Jin, H., & Kim, K. (2017, May). Hbm (high bandwidth memory) dram technology and architecture. In 2017 IEEE International Memory Workshop (IMW) (pp. 1-4). IEEE.

[5] Pérez, B.; Fell, A.; Davis, J.D. Coyote: an open source simulation tool to enable RISC-V in HPC. A: Design, Automation and Test in Europe Conference and Exhibition. "2021 Design, Automation & Test in Europe Conference & Exhibition (DATE): Grenoble, France, 1-5 February 2021: proceedings". Institute of Electrical and Electronics Engineers (IEEE), 2021, p. 130-135. ISBN 978-3-9819263-5-4. DOI 10.23919/DATE51398.2021.9474080.

PRACE SoHPCProject Title

Analysis of data management policies in HPC architectures

PRACE SoHPCSite

Barcelona Supercomputing Center, Spain

PRACE SoHPCAuthors

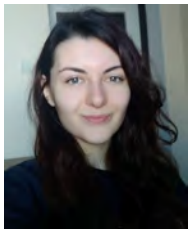
Aneta Ivaničová, Slovakia

PRACE SoHPCMentor

Borja Pérez, BSC, Spain

PRACE SoHPCContact

Aneta, Ivaničová, Matej Bel University in Banská Bystrica
Phone: +421 918 962 114
E-mail:
aivanicova@student.umb.sk



Aneta Ivaničová

PRACE SoHPCSoftware applied

Coyote

PRACE SoHPCMore Information

github.com/borja-perez/Coyote

PRACE SoHPCAcknowledgement

I would like to express my gratitude to my project partner Regina M. Gachomba, to our mentor Borja Pérez, to our co-mentor Teresa G. Cervero, to Rahul Shrivastava, to Alexander Fell, and to the whole MEEP team at the Barcelona Supercomputing Center.

PRACE SoHPCProject ID

2101

Coyote: A peek into the future of RISC-V Super-computers

Regina Mumbi Gachomba

Handling sparse workloads in HPC architectures can be a bummer, but Coyote, a performance modelling system and emulation platform, is here to the rescue! It proposes and analyses the viability of a decoupled architecture (ACME) in which the memory operations are separated from the computation.

Picture this: you want to revamp your old desktop computer to include new or updated features that will optimize its performance. Lucky enough, your system has a PCI Express bus; an expansion slot to which you can add graphic cards, drives etc. However, there is a threshold to how much you can reconfigure your system because some components are heavily dependent on the software. Furthermore, these components that you will add to your system will probably cost a lot, and if you don't know how to go about the reconfiguration, you will need to pay for the service as well. Now imagine doing this for a supercomputer (a system that usually fits a room. . . or two). It would take a while. . . and cost a fortune!

Nowadays, the flexibility of software-hardware co-design is a highly sought-after feature in most computing systems. The need for this flexibility spans across speed, scale, cost, innovation and so much more. To address this, Mare Nostrum Experimental Exascale Platform (MEEP) proposes a flexible FPGA based emulation platform, designed to explore hardware-software co-designs for future RISC-V supercomputers.[1] It would be easier to think of MEEP as a prototype that can be used to test the viability of a certain framework or architecture.

To demonstrate MEEP's emulation capabilities, an accelerator architecture called the Accelerated Compute and Memory Engine (ACME) will be incorporated into MEEP. ACME employs a decoupled architecture, which offers a clean-cut approach to how sparse and dense workloads are handled in computation. While dense HPC workloads are compute-bound, sparse workloads are memory-bound, as the vector elements need to be gathered/scattered using multiple requests. Data or workloads are considered sparse when certain expected values in a dataset are missing, which is a common phenomenon in the domain of High-Performance Computing (HPC) and High-Performance Data Analytics (HPDA). In addition, this data needs to arrive in time to the required destination. To achieve this in MEEP, a tool that can run and test various data movement and marshalling policies to decide on the optimal one, based on performance, is needed. Enter Coyote.

Coyote is a new open source, execution-driven simulation tool, that is capable of performance modelling and analysis of the ACME architecture before it can be cast onto silicon.[1] Coyote is founded on existing simulators (Spike and Sparta) and is being improved to cater for their shortcomings, especially in the High Performance Computing domain where the number of resources to be simulated is high, hence making it a powerful modelling tool. MEEP and coyote is a reference to the cartoon, Wile E. Coyote and the roadrunner, where Coyote aspires to match the roadrunner's using his wits. This

is precisely the goal of our simulator: To identify the best data movement and reordering policies that ensure a better leveraged access pattern for applications that will ensure faster and more efficient computing.

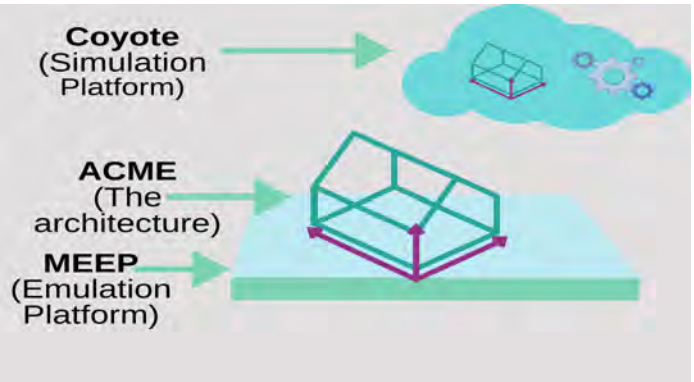


Figure 1: The fantastic three

Introduction – The Memory Tile

During the tenure of this internship, my work has been focused on setting up the basic functionality of the memory tile on the Coyote simulator. The memory tile houses the MCPU (Memory CPU), which can be loosely described as the 'intelligence' of the tile, responsible for organizing resources that are needed to perform the different memory operations.

These resources are obtained from the microengine, the vector address generator (VAG) and within the MCPU itself. The microengine is responsible for generating transactions for the instructions, whereas the vector address generator generates the memory requests. Another impressive feature of this memory tile is that it allows the re-usability of some already implemented functionalities. For example, a scalar load operation is handled like a unit stride vector load with a loop iteration of 1.

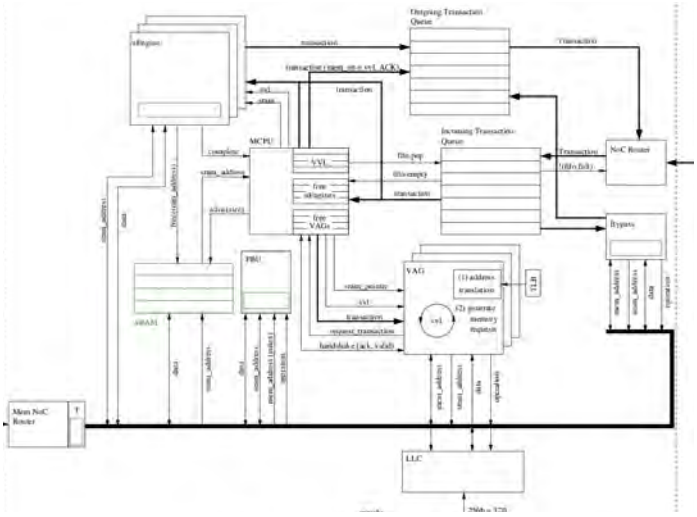


Figure 2: The architecture of the memory tile

Objectives

The primary objective was to understand how instructions, commands and data packets are to be received into the memory tile. Coyote allows us to create endless possibilities, even unrealistic ones, so it was important to also define the hardware constraints in the beginning so that we could obtain realistic results.

Once an overall understanding of the architecture was established, our goal was to simulate the different load and store operations and analyse the output and performance.[3] The types of scalar and vector operations to be simulated are as follows:

Unit stride: for vector elements that are located/stored next to each other in memory

Non-unit stride: for vector elements that are accessed at regular intervals, e.g., every second or third element

Indexed: for vector elements that are accessed by their indexed address. Quite similar to non-unit stride

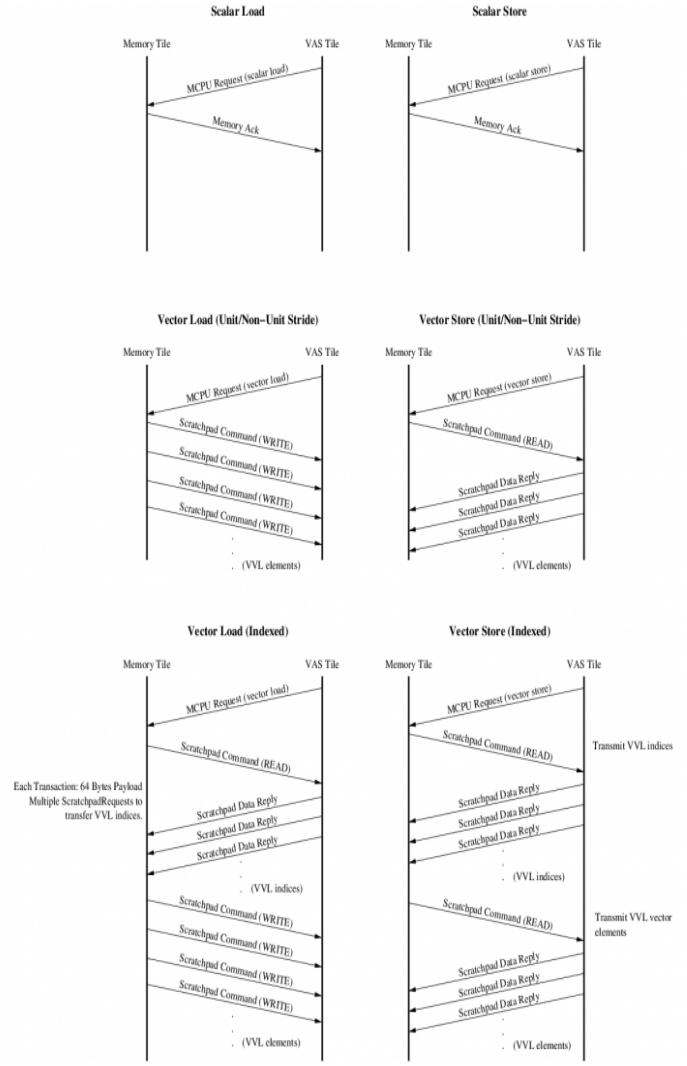


Figure 3: Communication sequences for various scalar and vector operations

Technical Work and Timeline

The first task involved setting up the bypass for scalar memory operations that do not need any resources from the MCPU. The scalar memory operations are handled as cache requests that are forwarded to the memory controller though the bus queue.

The MCPU functionalities that would cater for the load and store of vector operations were then set up. This entailed the handle function, controller cycles and the queues.

The Memory Tile and the VAS tile communicate using NoC messages. When these messages arrive at the memory tile, they can have either of the four payloads namely,

MCPU Instruction: vector memory operations

MCPUsetVVL: instruction to set virtual vector length (VVL) and sends it back to VAS tile

Scratchpad Requests: Commands such as free, allocate, read, write for the scratchpad

Cache Requests scalar memory operations and memory requests going to the MC

Each of these payloads are handled differently. For that reason, we use an overloaded function called `handle` in the source code. The handle function determines the queue and the controller cycle that will schedule the operation.

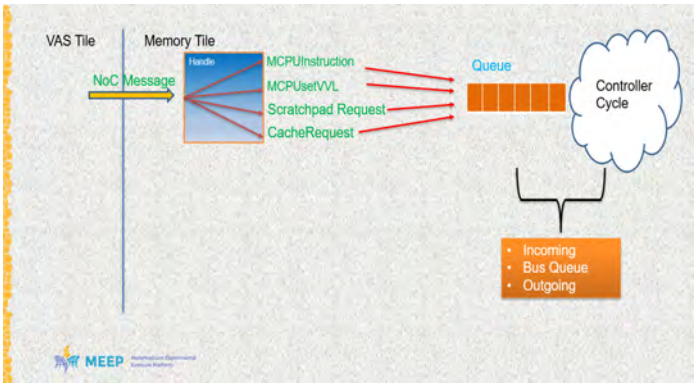


Figure 4: An illustration of how the overloaded function `handle` works

The controller cycle for incoming transactions essentially describes what happens in the MCPU. If an MCPU instruction is peeked from the queue, we first determine whether it is a load or a store. If it is a load, a scratchpad request to allocate some space in the scratchpad is created and sent back to tile (refer to figure 3). If it is a store, then a scratchpad command to load from the scratchpad is created. If the memory operation at the front of the incoming happens to be a scratchpad reply, it means that we had earlier sent a request to the scratchpad , and it is an indication that the subsequent computations can now be carried out.

The controller cycle for memory requests going through the bus queue schedules memory requests in the form of cache requests that will be forwarded to the Memory Controller, while the controller cycle for outgoing transactions schedules outgoing transactions from the MCPU, bypass and microengine.

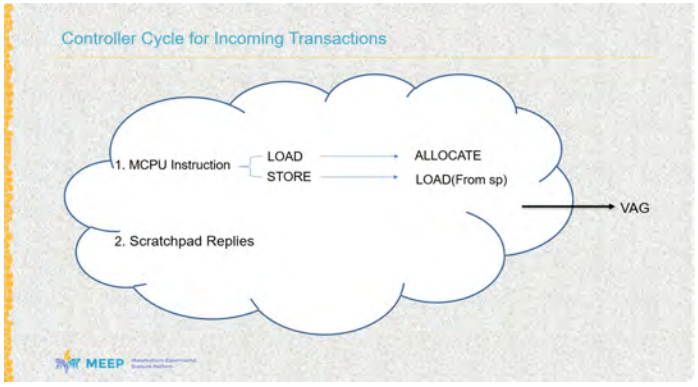


Figure 5: Controller cycle for incoming transactions

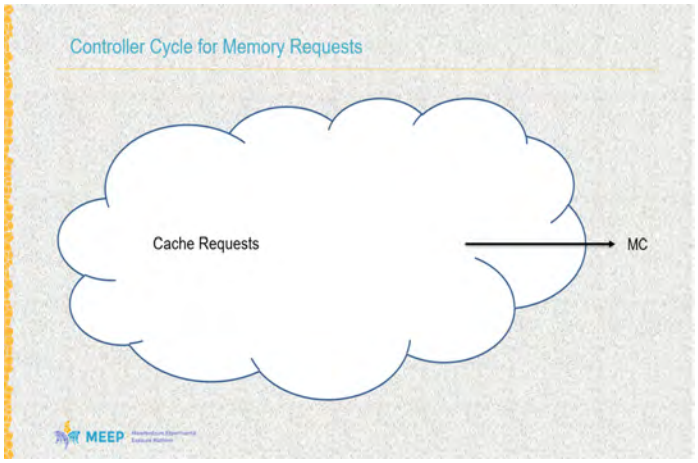


Figure 6: Controller cycle for memory requests generated by the VAG and sent to the memory controller

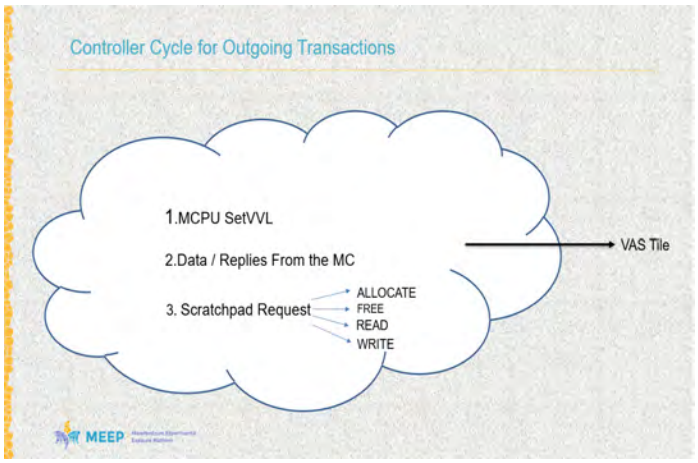


Figure 7: Controller cycle for outgoing transactions

A template class with the basic methods of a queue and boolean values that check the availability of the controller cycles was created. All the queues implement this template class, which is in the form of a header file (`Bus.hpp`). The reason for this was to reduce code replication of the push and pop functions of all the queues in the memory tile.

In addition, The MCPUwrapper is initialised with a hash map (unordered map) to keep track of the cache requests and scratchpad requests that are generated from each MCPU Instruction. The MCPU instructions are consequently initialised with an ID parameter that we use as the key in the hash map.

Results

The Coyote simulator can now carry out both scalar vector load and store operations in the memory tile. Address, data, and control packets can be sent to the accelerator tile and are received in the memory tile. However, there is still some debugging to be done. For instance, when scratchpad requests return from the memory controller, they do not arrive in the same order they were sent. Although the simulator has control of how memory operations are scheduled in the queues, it is still not well-defined how the operations are ordered when they are in the MC, MCPU or the microengine.[5]

```
93489   Core 0 simulated 72968 instructions
93490   Core 1 simulated 70447 instructions
93491   Core 2 simulated 70473 instructions
93492   Core 3 simulated 70456 instructions
93493   Total simulated instructions 284344
93494   The monitored section took 0 nanoseconds
```

Figure 8: This is the number of simulations that were run when coyote was run with four cores

```
The cmd is ../../apps/mt-matmul-vec/matmul
1
Calling sim with smart mcpu 1
Created
2
3
48: memory_cpu: receiveMessage_noc_: Received from NoC: Src: 0, Dest: 0, Type: 1
[0;36m48: memory_cpu: handle CacheRequest: 0x1000 @ 1, coreID: 0x0[0;0m
31: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x1000 @ 1, coreID: 0x0
109: memory_cpu: Returned from MC: instrID: 0 - CacheRequest using the Bypass: 0x1000 @ 1, coreID: 0x0
[0;32m10a: memory_cpu: controllerCycle_outgoing_transaction: Sending to NoC: Src: 0, Dest: 0, Type: 5[0;0m
```

Figure 9: Cache request transaction

```
325  Issuing MCPU VVL from core 0 and tile 0
326  Issuing MCPU VVL from core 1 and tile 0
327  Issuing MCPU VVL from core 2 and tile 0
328  Issuing MCPU VVL from core 3 and tile 0
329  1406: memory_cpu: receiveMessage_noc_: Received from NoC: Src: 0, Dest: 0, Type: 6
330  [1;36m1406: memory_cpu: handle MCPUSetVVL: AVL: 20, VVL: 20, lmul: 1, w: 8, coreID: 0[0;0m
331  1407: memory_cpu: receiveMessage_noc_: Received from NoC: Src: 0, Dest: 0, Type: 6
332  [1;36m1407: memory_cpu: handle MCPUSetVVL: AVL: 20, VVL: 20, lmul: 1, w: 8, coreID: 1[0;0m
333  [0;32m1407: memory_cpu: controllerCycle_outgoing_transaction: Sending to NoC: Src: 0, Dest: 0, Type: 6[0;0m
334  1408: memory_cpu: receiveMessage_noc_: Received from NoC: Src: 0, Dest: 0, Type: 6
335  [1;36m1408: memory_cpu: handle MCPUSetVVL: AVL: 20, VVL: 20, lmul: 1, w: 8, coreID: 2[0;0m
336  [0;32m1408: memory_cpu: controllerCycle_outgoing_transaction: Sending to NoC: Src: 0, Dest: 0, Type: 6[0;0m
337  1409: memory_cpu: receiveMessage_noc_: Received from NoC: Src: 0, Dest: 0, Type: 6
338  [1;36m1409: memory_cpu: handle MCPUSetVVL: AVL: 20, VVL: 20, lmul: 1, w: 8, coreID: 3[0;0m
339  [0;32m1409: memory_cpu: controllerCycle_outgoing_transaction: Sending to NoC: Src: 0, Dest: 0, Type: 6[0;0m
340  [0;32m140a: memory_cpu: controllerCycle_outgoing_transaction: Sending to NoC: Src: 0, Dest: 0, Type: 6[0;0m
```

Figure 10: setVVL transaction

```
412  198e: memory_cpu: controllerCycle_incoming_transaction: 0x80042e00 @ 1978 Op: 0x0 SubOp: 0x0, width: 0x8, coreID: 0x2,
413  198e: memory_cpu: memOp_unit: noepr: 8, re: 20, address: 80042e00
414  198e: memory_cpu: memOp_unit: noepr: 8, re: 18, address: 80042e40
415  198e: memory_cpu: memOp_unit: noepr: 8, re: 10, address: 80042e80
416  198e: memory_cpu: memOp_unit: noepr: 8, re: 8, address: 80042ec0
417  198f: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042c80 @ 1978, coreID: 0x0
418  198f: memory_cpu: controllerCycle_incoming_transaction: 0x80042f00 @ 1978 Op: 0x0 SubOp: 0x0, width: 0x8, coreID: 0x3,
419  198f: memory_cpu: memOp_unit: noepr: 8, re: 20, address: 80042f00
420  198f: memory_cpu: memOp_unit: noepr: 8, re: 18, address: 80042f40
421  198f: memory_cpu: memOp_unit: noepr: 8, re: 10, address: 80042f80
422  198f: memory_cpu: memOp_unit: noepr: 8, re: 8, address: 80042fc0
423  1990: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042cc0 @ 1978, coreID: 0x0
424  1991: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042d00 @ 1978, coreID: 0x1
425  1992: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042d40 @ 1978, coreID: 0x1
426  1993: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042d80 @ 1978, coreID: 0x1
427  1994: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042dc0 @ 1978, coreID: 0x1
428  1995: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042e00 @ 1978, coreID: 0x2
429  1996: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042e40 @ 1978, coreID: 0x2
430  1997: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042e80 @ 1978, coreID: 0x2
431  1998: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042ec0 @ 1978, coreID: 0x2
432  1999: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042f00 @ 1978, coreID: 0x3
433  199a: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042f40 @ 1978, coreID: 0x3
434  199b: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042f80 @ 1978, coreID: 0x3
435  199c: memory_cpu: controllerCycle_mem_request: Sending to MC: 0x80042fc0 @ 1978, coreID: 0x3
```

Figure 11: Unit-stride vector transaction in which the data for a vector load is returned in multiple NoC transactions.

Future Work

At the moment, the analysis of Coyote's performance is done mostly on the command line. This is expected to shift to the use of a visualization tool, Paraver, which is a flexible data browser developed at BSC, used to capture the behaviour of parallel programs, and give a quantitative analysis of the problem.[4] Due to its flexibility, it perfectly meets the needs of Coyote in testing the large number of data management policies with different workloads used in HPC.

References

[1] Fell, A., Mazure, D. J., Garcia, T. C., Perez, B., Teruel, X., Wilson, P., & Davis, J. D. (2021). The MareNostrum Experimental Exascale Platform (MEEP). Supercomputing Frontiers and Innovations, 8(1), 62-81

[2] Perez, B., Fell, A., & Davis, J. D. (2021, February). Coyote: An Open Source Simulation Tool to Enable RISC-V in HPC. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 130-135).

[3] MEEP Wiki https://wiki.meep-project.eu/index.php/The_Coyote_simulator

[4] Paraver: a flexible performance analysis tool https://tools.bsc.es/paraver

[5] GitLab repository https://gitlab.bsc.es/meep/meep-performance-modelling/coyote-tool/coyote-sim/-/tree/basicVersionMemoryTile

PRACE SoHPCProject Title

Analysis of Data Management Policies in HPC architectures

Regina Mumbi Gachomba

PRACE SoHPCSite

Barcelona Supercomputing Center, Spain

PRACE SoHPCAuthors

Regina Mumbi Gachomba, Ankara Yildirim Beyazit University, Turkey

PRACE SoHPCMentors

Borja Perez, Barcelona Supercomputing Center, Spain
Alexander Fell, Barcelona Supercomputing Center, Spain

PRACE SoHPCContact

Regina Mumbi Gachomba, Ankara Yildirim Beyazit University
Phone: +90 553 7108223
E-mail: mumbigachomba254@gmail.com

PRACE SoHPCSoftware applied

C++, GitLab, Spike, Sparta, Coyote

PRACE SoHPCMore Information

https://github.com/borja-perez/Coyote

PRACE SoHPCAcknowledgement

I would like to express immense gratitude to my mentors Alexander Fell, Rahul Shrivastava, Borja Perez and Teresa Cervero for their valuable guidance through my tasks in the project. I also had the great pleasure of working with my project partner, Aneta Ivaničová and the whole MEEP team at BSC. Thanks to PRACE for organising this valuable programme.

PRACE SoHPCProject ID

2101

Tungsten Simulations for Nuclear Fusion

Paolo Settembri and Eoin Kearney

Now that fusion power is a closer reality, with several large scale reactor experiments performed around the world, it is increasingly important to expand our knowledge in the fields related to this subject. One of these is materials science, focusing on materials used in the fusion reactors (e.g. tungsten), and the damage that the fusion products cause to them. Investigating these structures and damages is a hot topic in material science.

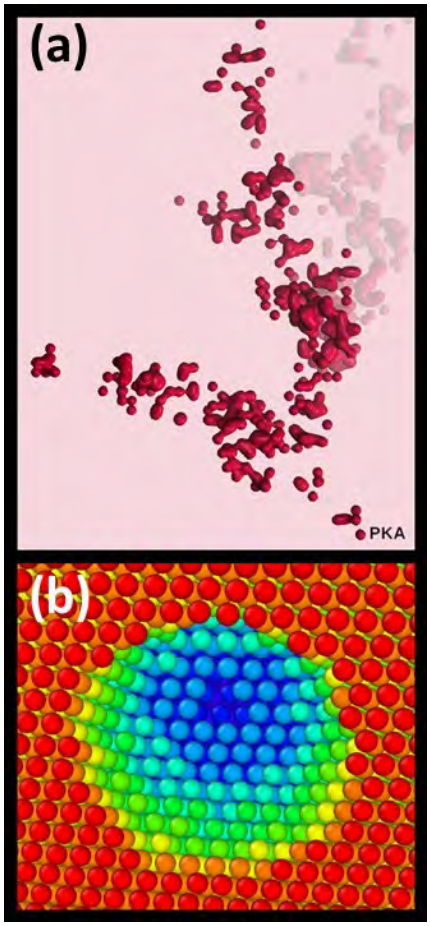


Figure 1: (a) Tungsten cascade simulation with all atoms transparent except those displaced. The PKA=Primary Knock-on atom, where the cascade begins. (b) Example of an empty bubble inside a tungsten perfect crystal (coloring for depth's understanding)

The search for nuclear fusion has spanned more than 50 years, with potential to be the green energy source of the future. The most viable process for industrial fusion reactors is the deuterium-tritium reaction[1], in which two isotopes of hydrogen are brought together under sufficient force to fuse them inside a nuclear fusion reactor, releasing excess energy. Figure 2 shows a sketch of this, with helium atoms, neutrons and energy being the products of the collision between deuterium and tritium. Currently, fusion for energy generation purposes has been achieved, but consumes more energy than it releases. However, new, more ambitious developments such as the test reactors ITER in the EU and JET in the UK, suggest progress to a viable reactor is growing closer. Given the extreme conditions involved (high temperature, and continuous neutron bombardment) extremely versatile materials are required to provide a long operating life. Tungsten metal is a likely candidate for these conditions, due to its stability, high melting point and good thermal conductivity[2]. This project will focus on the damage to its structure caused by neutron bombardment.

In the following sections we will present the motivation of our project in more detail and talk through the results of our studies.

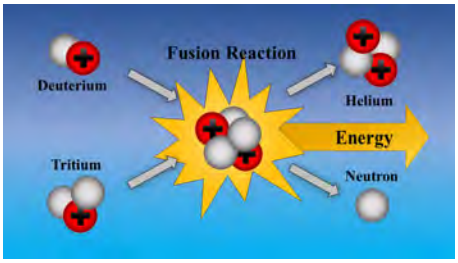


Figure 2: Sketch of the fusion reaction.

Motivation

Neutron bombardment can have several effects on a metal including electronic excitement, transmutation reactions, and kinetic displacements of atoms forming defect sites[2]. This report focuses on the latter. These defects form via a cascade, which is comparable to projectile motion; an atom with high energy following a neutron impact can shoot out of the crystal lattice structure, disrupting many other atoms around it, with the effects fanning out conically from the initial atom. This initial atom is known as a primary knock-on atom or PKA. The cover photo (Figure 1a) highlights atoms displaced in such a process. This process occurs at the atomic scale, but introduces weaknesses that have strong local effects on material performance, and which can build up over prolonged neutron bombardment forming larger vacancies. Moreover, hydro-

gen and helium byproducts of the fusion process can also accumulate in these defects and affect tungsten properties.[2] Analyzing these will be an area of interest for this project.

Additionally, neutron sources for materials testing at fusion relevant energies are not currently available, though the International Fusion Materials Irradiation Facility (IFMIF) is planned for construction to cover this, with Granada (Spain) as the proposed location. Computational simulations provide an insightful way to investigate material behavior under these conditions, to analyze and replicate these cascade effects, evaluate the materials' performance and identify potential improvements. Fusion reactors are subject to extreme heating, and so thermal conductivity is a key property to track and will be one of the focuses of this project. Thermal conductivity describes the way the system behaves under the effect of heat sources[1]. Hence it is fundamental to understanding how the system responds to steep temperature gradients, before and after the formation of defects in the structure and following insertion of helium and hydrogen inside the reactor[3]. To investigate, we used the atomistic modelling software LAMMPS to simulate tungsten metal; a defect cascade of tungsten is simulated to recreate a structure eroded by fusion, allowing tests of vacancy formation with focus on their effect on thermal conductivity. Our aim is to shine more light on the reaction of tungsten to neutron irradiation.

Methods

LAMMPS is a type of molecular dynamics modelling software. These are a form of computer simulation which represent species on the level of atoms that interact using Newton's law of motion and a given potential. These simulations have great performance and scalability which allows the simulation of even millions of atoms on just a few HPC nodes within a reasonable time[2]. E.g. in this project a two million atom cascade took two hours to run on ten MareNostrum-4 nodes. This allows evaluation of both defect formation and behavior under both neutron bombardment[1]. In the simulations a parallel-piped simulation box is defined and tungsten atoms are placed in a body centered cubic (BCC) crystalline structure, which is tungsten's crystalline structure; then an equilibration run is performed at a given system temperature. We used 300K for thermal conductivity to compare the results with previous papers, and 1K for defect cascades to minimize thermal motion during procedure development. To have this constant temperature system LAMMPS uses a modified version of the equation of motion, this procedure is called the Nose-Hoover algorithm. The simulations were equilibrated initially to reach the set temperatures.

Defect Cascades

To start with, a procedure was prepared for defect cascades in LAMMPS. This uses a common format for cascades in which an internal region of a tungsten cube is designated for the cascade, with an outer area lining it.
The purpose of this is twofold. Firstly the simulation box must be large enough to contain the cascades, as otherwise cascade atoms could cross the periodic boundary and affect atoms around the PKA site. The two regions mark the site of concern, allowing an area of one full cascade to be examined. Secondly this allows two thermodynamic regimes to be implemented; the inner cascade region with variable temperature and the outer border region with temperature control. This is necessary as thermostats modify the velocity and so must be avoided in the cascade region to avoid a loss of accuracy, while the outer region acts like a heat bath.

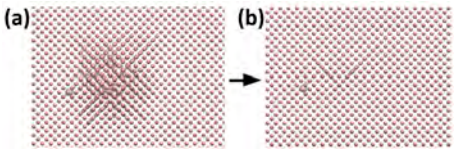


Figure 3: Example of a defect cascade in tungsten. (a) Cascade wave. (b) Final structure. The large atom on the left side of each is the PKA.

With this set up an atom was chosen for the PKA and assigned an energy, which will be varied for analysis. The size of the simulation was modified to allow for the larger energy simulations large scale cascades. The number of defects formed was then calculated using the visualization tool Ovito; specifically its Wigner-Seitz defect analysis tool. Figure 3 shows a cascade in two steps. Initially, (3a), a wave passes through the structure causing many displacements most of which settle back into position. Hence, (3b), the final stage has fewer displacements. This reflects the trend found in previous literature[4].

Table 1: Number of final defect sites remaining after 200 keV cascade.

Potential	Number of Defects
EAM1[5]	124
EAM2[5]	136
EAM3[6]	124
WTa[7]	351
WRe[8]	105
WRe Set[9]	236

Simulations were successful with energies up to 200 keV and 2 million atoms, as well as with a variety of tungsten potentials. The number of permanent defects formed by each potential considered is given by table 1.

Thermal Conductivity

To calculate the thermal conductivity of pure tungsten metal and study its variations with the presence of vacancies, various molecular dynamics simulations have been performed. The procedure used and the obtained results are detailed in the following.

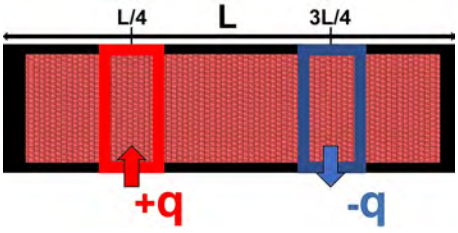


Figure 4: Sketch of the simulation box with the positive and negative heat fluxes.

Two regions are defined along one direction of the simulation box, of equal volume and equidistant from the box borders. We then start heating one region with a positive heat flux $\mathbf{q}$, while cooling the other with an equal negative heat flux $-\mathbf{q}$. A sketch of this system can be seen in figure 4. In this way the average temperature of the system stays constant, but the heated region will have a higher local temperature while the cooled region will have a lower one. We then have a temperature gradient ∇T between these regions; knowing the value of the heat flux $\mathbf{q}$ and calculating ∇T, we can obtain the value for the thermal conductivity k from Fourier's law $\mathbf{q}$=-$k\nabla$T. This procedure has been performed using different potentials. The results are shown in Table 2.

Table 2: Thermal conductivity of Tungsten

Potential	k
Ackland[11]	38.60 W/mK
WTa[7]	17.03 W/mK

These results have been compared and found to be compatible with the values from previous works: from MD studies[12,13,14] it was found that k=15-21 W/mK while an ab-initio paper[15] obtained k=46 W/mK. We then added an empty bubble at the center of the system with radius r, like the one in Figure 1b, resembling the damage caused by neutron bombardment, and repeated the same procedure obtaining a new result for the thermal conductivity of the system. Using the Ackland potential[11] we investigated the relative variation of the thermal conductivity with respect to the value for a perfect crystal increasing the size of the empty sphere at the box center.

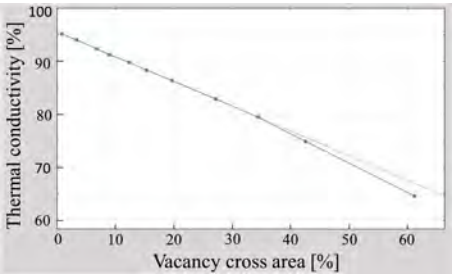


Figure 5: Thermal conductivity variation as a function of the sphere cross area: data points and expected linear behaviour

The obtained result is shown in figure 5. We can see a decreasing linear behavior in thermal conductivity as a function of the transverse (with respect to the heat flux direction) sphere area πr^2, in line with previous results.[14] For high radius spheres we start to see the effect of the finite size of the simulation box; the data shift away from the expected behavior. We can see in a qualitative way how much the presence of such a vacancy effects the system's thermal conductivity.

Conclusion

In this project defect cascades were successfully performed. Our results, shown in Table 1, present differing numbers of defects formed for several potentials. All the potentials used show more defects then a recent similar study (which reported 102 defects at 300K).[9] However, another study[10] investigating tungsten cascades concluded that higher temperatures show increasing defect recombination rates. As our cascades were performed at 1K, some larger values are expected. Most of the potentials are still near the literature value, with the WRe potential coming closest.
The difference in the number of defects formed depends on the system the potentials were created to represent: EAM1 and EAM2 represent H and He interactions with tungsten; EAM3 gives a general representation of tungsten-tungsten interactions: WRe, WReSet and WTa represent two tungsten alloys. Further work arising from this project would include larger cascades as interaction between the inner and outer thermodynamic regions may impact the WReSet and WTa potentials performance. Identifying solutions will allow integration of more potentials to the procedure allowing analysis of more complex structures e.g. doped tungsten. The lining up of number of defects with the majority of the tested potentials, and these settling near literature values confirms the correctness of the procedure used for defect cascades. In the part of the project focused on the calculation of the thermal conductivity of tungsten, results compatible with previous works have been achieved for a perfect crystal. These show a dependency upon the used potential for the same reasons as explained above.
Vacancies have also been introduced in the structure and the their effect on thermal conductivity has been studied, showing a linear decrease with the vacancy cross sectional area, as it has been found in previous papers.
To give an idea of how important these effects are, removing 11 atoms out of half a million atoms system decreases the thermal conductivity by 5%. Our results can be the basis for future works centered on the effect of the presence of Helium and Hydrogen atoms inside the vacancies and the evaluation of the effects of a defect cascade on the system thermal conductivity.

References

[1]R. Abernethy, Mater. Sci. Technol., 2016,33, 388-399
[2]M. Gilbert et al., J. Nucl. Mater., 2021, 554, 153113
[3]J. Marian et al., Nucl. Fusion, 2017, 57, 092008
[4]M. Rajput et al, Fusion Eng. Des., 2020, 150, 111370
[5]G. Bonny et al, J. Phys. Condens. Matter, 2014, 26, 485001
[6]M. Marinica et al, J. Phys. Condens. Matter, 2013, 25, 395502
[7]Y. Chen et al, Comp. Mat. Sci., 2019, 163, 91-99
[8]G. Bonny et al, J. App. Phys., 2017, 121, 165107
[9]W. Setyawan et al., J. Appl. Phys., 2018, 123, 205102
[10]W. Setyawan et al., J. Nucl. Mater., 2015, 462, 329-337
[11]S. Han et al., J. Appl. Phys., 2003, 93, 3328-3335
[12]L. Hu et al., Appl. Phys. Lett., 2017, 111, 081902
[13]B. Fu et al., J. Nucl. Mater., 2012, 427, 268-273
[14]H. Zhang et al., Fusion Eng. Des., 2020, 161, 112004
[15]Y. Chen et al., Phys. Rev. B, 2019, 99

Acknowledgements

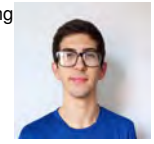
We'd like to thank Julio Gutierrez for his patience and great supervision, the BSC team and especially the fusion group, managed by Mervi Mantsinen. We are very thankful to all the SoHPC organizers, for going above and beyond to make the program happen. This project was developed within the framework of the FusionCAT project (001-P-001722) co-financed by the ERDF Operational Program of Catalonia. Part of the simulations were carried out using supercomputer resources provided under the EU-JA Broader Approach collaboration in the Computational Simulation Centre of International Fusion Energy Research Centre (IFERC-CSC).

PRACE SoHPC Project Title
Computational atomic-scale modelling of materials for fusion reactors

PRACE SoHPC Site
Barcelona Supercomputing Center

PRACE SoHPC Authors

Paolo Settembri and Eoin Kearney,
University of L'Aquila, Italy
University of Edinburgh, UK


Paolo Settembri


Eoin Kearney

PRACE SoHPC Mentors
Julio Gutiérrez Moreno, BSC, Spain
Mervi Mantsinen, BSC, Spain

PRACE SoHPC Contact

Eoin Kearney, University of Edinburgh
E-mail: ekearney97@gmail.com
Paolo Settembri, University of L'Aquila
E-mail: paolosettembri@libero.it

PRACE SoHPC Software applied
LAMMPS, Ovito, VMD

PRACE SoHPC More Information
BSC Fusion Group

PRACE SoHPC Project ID
2102

Improving the Performance of Fault Tolerance

Athanasios KASTORAS & Kevser İLDEŞ

High Performance Computing systems' complexity has increased the possibility of system failures. This project's goal was to improve the checkpoint/restart technique's performance using two different methods, lossy compression and precision bound differential checkpointing.

Reliability is a big issue and it is more significant especially in supercomputers. In High performance computing (HPC), systems are built from highly reliable components but with the increase in the number of components, the likelihood of failure becomes a serious issue as the overall failure rate of supercomputers also increases. Nowadays, the mean time between failures (MTBF) of a large-scale HPC system is about a day which means that approximately once a day the system faces a failure.

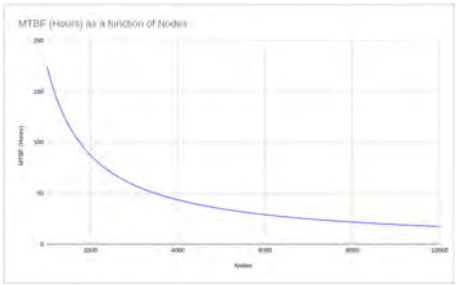


Figure 1: MTBF.

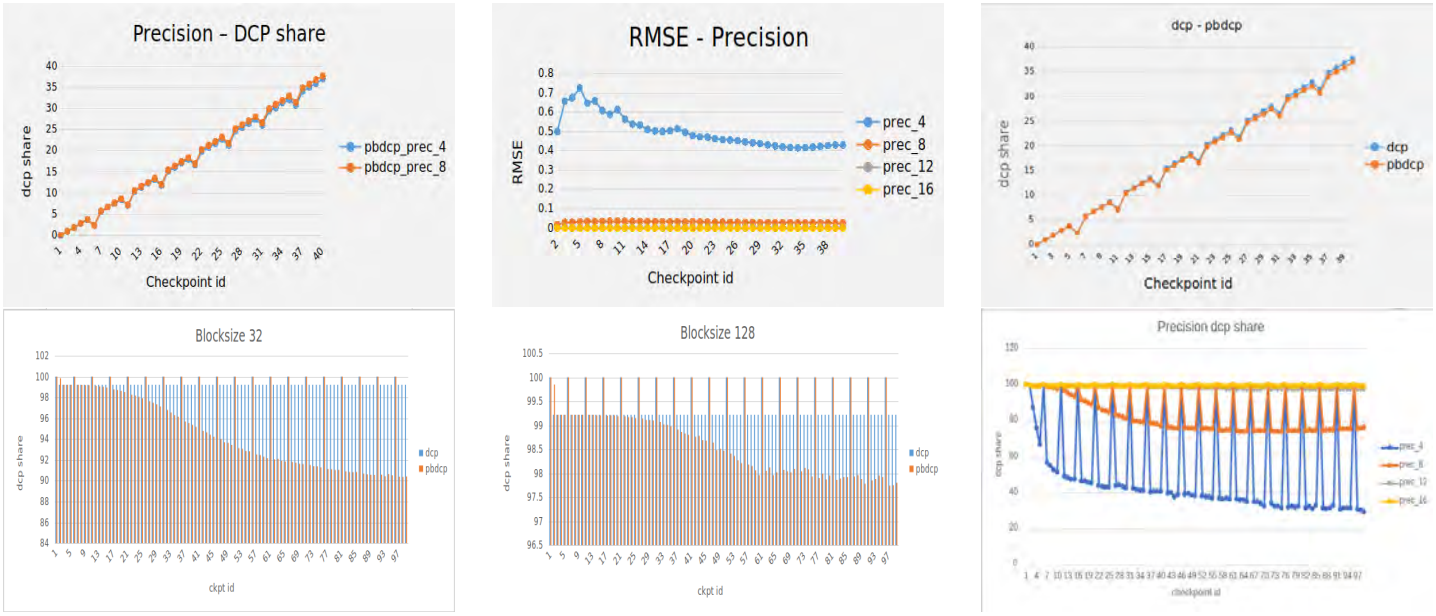
Of course, all processes running during the failure will be killed and their data will be lost. This can be devastating on large-scale applications such as complex numerical simulations which may execute for days, weeks, or even months. Therefore, the development of Fault Tolerance mechanisms is necessary to be able to protect the application and continue improving supercomputers' performance.

There are several techniques to protect the application and checkpoint-and-restart is a common one. It means taking snapshots of the application at specific times which means saving the system state in stable storage, frequently a parallel file system (PFS), and in case of a failure restarting the execution from the last recovery point. The disadvantage of this method is that in large-scale applications there is an enormous number of data to be stored, and due to the file system's bandwidth restrictions there is a high risk of creating a bottleneck and dramatically increase the processing time. There are some advanced techniques to fight such limitations like multilevel and differential checkpointing.

Fault Tolerance Interface (FTI) is an application-level checkpointing Library which allows users to protect selected datasets using the Checkpoint/Restart method. There are different checkpoint levels which are:

- Level 1: Local checkpointing on the nodes. Fast and efficient against soft and transient errors.
- Level 2: Local checkpointing on the nodes + copy to the neighbor node. Can tolerate any single node crash in the system.
- Level 3: Local checkpointing on the nodes + copy to the neighbor node + Reed Solomon encoding. Can tolerate correlated failures affecting multiple nodes.
- Level 4: Flush of the checkpoints to the Parallel File System (PFS). Tolerates catastrophic failures such as power failures.
- L4 dcp: Differential checkpointing.

In differential checkpointing, instead of storing all the data in every checkpoint, they are stored once and then only the differences are updated



Graphs obtained from experiments for Precision Bound Differential Checkpointing feature.

from the new data thus decreasing the quantity of data that are transferred.

In Figure 2 difference between without checkpointing in which in case of a failure there is a need of restarting from the beginning, classical and differential checkpointing, where it restarts from last recovery point is shown, of course checkpoint sizes differs in differential checkpointing (dcp).

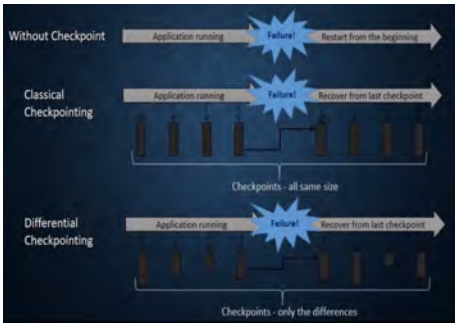


Figure 2: Difference between checkpointing approaches.

Throughout this project two new mechanisms are added to FTI library which are Precision Based Differential Checkpointing (PBDCP) and Compressed Checkpointing (CPC),using lossy compression.

Precision Based DCP (PBDCP)

Approximate computing promises high computational performance combined with low resource requirements like very low power and energy consumption. This goal is achieved by relaxing the strict requirements on accuracy and precision and allowing a deviating behavior from exact boolean specifications to a certain extent.

Floating point numbers map decimal numbers to a unique bit representation and they have 3 parts: sign bit, mantissa and exponent. The idea of Precision Based Differential Checkpointing is to cap the last bits of mantissa that are the least significant bits according to the given precision value. As an example in Figure 3, if the precision value given by user is sixteen, it truncates the last seven bits of mantissa, (i.e. makes them 0).

Therefore, PBDCP allows us to take benefits from dcp share which shows the difference between checkpoints, those would be stored, for such small changes and thus transferring less data.

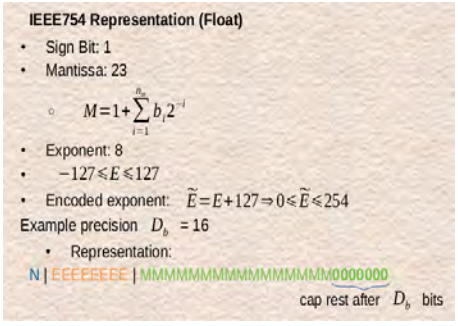


Figure 3: Example of pbdcp operation for ieee754 floating point representation.

Usage

To use this mechanism, there are some basic additions in the configuration file. Firstly, enable_pbdcp value should be set to 1 and pbdcp_precision value should be entered.

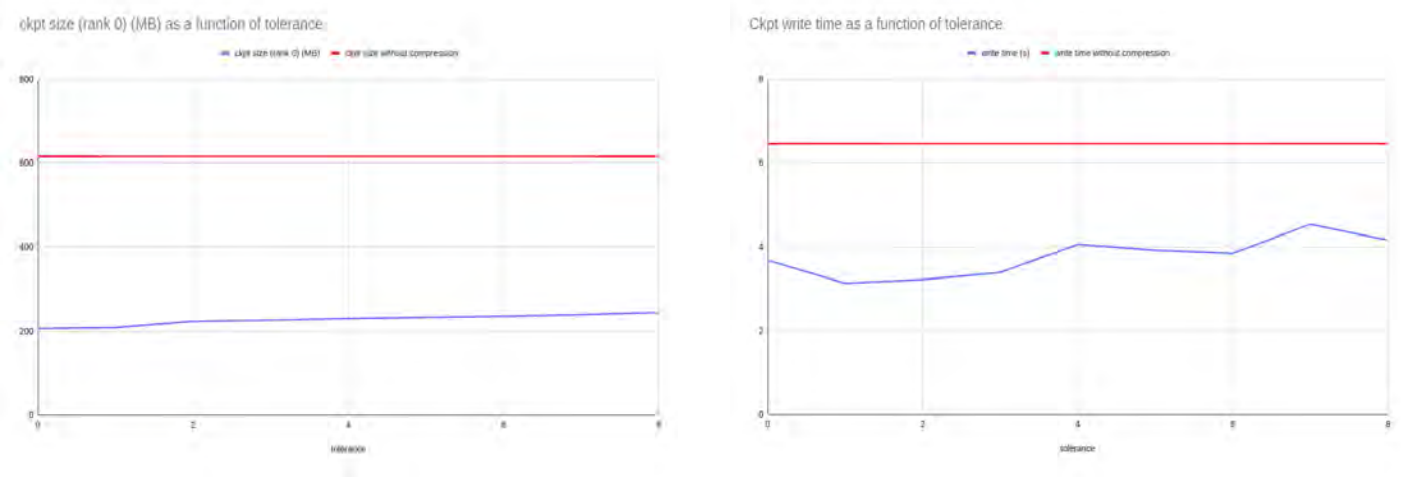
Then, in the application when calling FTI_Checkpoint function, user should specify the level as pbdcp.

Experimental Results

As expected, the smaller the precision the most benefit we can get from dcp share since the value remains unchanged for a certain interval but also the larger the rmse value which is route means square error for uncapped and capped values and it can also be seen in charts Precision against DCP share and RMSE on top on the page in which dcp blocksize: 1024, iteration: 200 and checkpoint interval: 5 (which means there are 40 checkpoints) values are used. And for comparing pure dcp and precision based dcp, again as expected dcp share is lower in pbdcp and an example result of an execution with values as Block size: 1024, Precision: 4, Iteration: 200, Ckpt Interval: 5, is shown in graph on the upper left corner.

Then, for experiments based on blocksizes with values as iteration: 5000, Interval: 50 and precision value is given as 12 in Blocksize graphs on top of the page. As a result, we can say that when blocksize increases, dcp share also increases but still we can get advantage of pbdcp.

Lastly comparing dcp share against precision keeping blocksize constant, 128 for this example, we can see in the chart above in larger precision values the change is smaller.



Graphs obtained from experiments for lossy compression feature. The left figure shows a graph of checkpoint size as a function of tolerance (blue) in comparison to the checkpoint size without compression (red). The right figure shows a graph of checkpoint write time as a function of tolerance (blue) in comparison with the write time without compression (red).

Checkpointing with Lossy Compression (CPC)

Lossy compression can be defined as the class of data encoding methods that uses inexact approximations and partial data discarding to represent the content. Our goal, was to use lossy compression to compress the checkpoints of level L4 before transferring them from the local node to the PFS. We can assume that the time needed for the compression to be made is very small in comparison to the time needed to transfer the data though the network, so we don't actually mind for the compression's performance as long as it has a high compression rate. We implemented lossy compression to the FTI using the zfp library.

Usage

To use the lossy compression feature, three extra parameters must be specified: compression_enabled, cpc_block_size, and cpc_tolerance. The first one is simply a boolean value which, when set to 1, allows the checkpoints to be compressed. The second one specifies the maximum memory size which can be allocated for the compression and is used to protect memory restricted systems. It is suggested that the block size is not significantly lower than the actual data size, because then it will interfere with the compression and may decrease the compression ratio. The last parameter defines the absolute error tolerance for the compression. The actual error tolerance is given by the function 10-t where t is the integer cpc_tolerance specified at the configuration file. The user must specify a tolerance suited for their application, high enough to maximize the compression rate, but low enough so the results won't be altered.

Experimental Results

Testing Accuracy

Figure 4: Precision in lossy compression

To test the accuracy of the compression we used a simple heat distribution application for which the sum of the squares of the absolute error can be seen at the image above.

Testing Performance

As a principle, lossy compression targets large-scale applications. Although the performance of lossy compression differs for different data sets, in programs with small memory, using lossy compression may even slow down checkpointing due to the time needed to perform the compression and the possibility of hard to compress data. Therefore, to test the application's performance we have to employ a very scalable application. For our experiments, we run LULESH on BSC's Marenostrum cluster, with a size of 615,85 Megabytes per process, for 512 processes in 16 different nodes, using different values for cpc_tolerance. The experiment can be considered successful since a measurable decrease in the time to write the checkpoint was observed. As you can see in figures 1 and 2, the write times of the checkpoints for any value of tolerance between zero and eight seem to be around half the write time when the checkpoint isn't compressed. Also, we can see that tolerance doesn't really affect write time, since for different tolerances we don't observe high differences at the checkpoint file. This may change in other experiments with different data consistency or size.

PRACE SoHPCProject Title

Precision based differential checkpointing for HPC applications

PRACE SoHPCSite

Barcelona Supercomputing Center, Spain

PRACE SoHPCAuthors

Kevser İLDEŞ, Turkey
Athanasios KASTORAS, Greece


Kevser İLDEŞ


Athanasios KASTORAS

PRACE SoHPCMentor

Kai Keller, BSC, Spain
PhD. Leonardo Bautista Gomez, BSC, Spain

PRACE SoHPCContact

Kevser İLDEŞ, Marmara University
E-mail: kevserildes@gmail.com
Athanasios KASTORAS, University of Thessaly
E-mail: akastoras@uth.gr

PRACE SoHPCSoftware applied

ZFP library
FTI library

PRACE SoHPCMore Information

https://summerofhpc.prace-ri.eu

PRACE SoHPCAcknowledgment

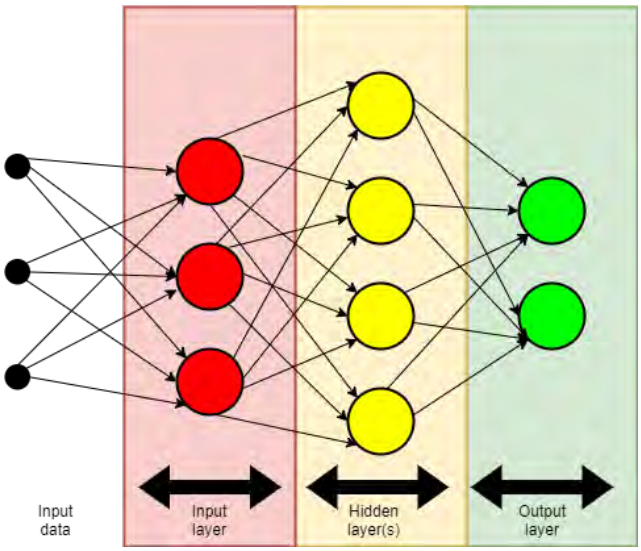
We are grateful to our mentors Kai Keller and Leonardo Bautista Gomez for their continuous help during this project, to the BSC education team for the organization of the internship and PRACE for making SoHPC possible.

PRACE SoHPCProject ID

2103

ML From HPC to the Edge

Jakub Raczyński, Mehmet Enes Erciyes



ML applications require enormous compute resources to train. However, they should not be limited to uses with only high end devices, but allow edge devices with limited resources to be used for them. Our project aimed to research ways of optimizing big models for resource-constrained environments. We found how pruning worked for big models and used it in an example case of object detection with a Raspberry Pi.

Machine learning applications, especially deep learning, have been very ubiquitous in the past few years. Deep learning is a subset of machine learning which uses neural networks that have lots of layers, thus deep. Neural networks outperform almost any other model in many domains, however, they require lots of data and lots of compute resources to be trained. The amount of data required may easily find hundreds of gigabytes, while state of the art models require days of training on multiple GPUs. This makes HPC a common tool for training neural networks. After the training, models are usually deployed to a cloud environment with high-end resources and provide inferences over APIs. However, sometimes, it is much better to directly run the inference on the edge devices, i.e. smartphones, development boards, daily-use laptops etc. In this project, we looked into ways of training neural networks on HPC clusters and then pruning them to get speed-ups and lower the space requirement. Then, we tried to implement an example case with object detection on a Raspberry Pi.

What are neural networks?

But first, what are neural networks? Nowadays, they are the most popular and efficient AI models, especially in computer vision and natural language processing. They are loosely inspired by the human brain. They consist of nodes called neurons and the connections between them called activations. Training data changes the weights of the activations, so that, when an input is fed to the neural network, it produces a desired outcome. The neurons create layers. A deep neural network usually has more than 20 layers.

Convolutional Neural Networks

In this project, we wanted to focus on computer vision applications. Therefore, we used convolutional neural networks which are specialized neural networks for image tasks. Standard neural networks are called fully connected networks. However, since images have lots of pixels, a fully connected network would have too many weights and biases, thus increasing the training time and making it harder for the network

to learn good representations. Convolutional neural networks use filters and use the convolution operation to learn features from data and decrease the parameter numbers to a reasonable level.

Training image classification models on HPC

Image classification is the task of identifying which class does the given image belong to among a set of classes. One of the most important datasets for this task is ImageNet dataset. The ImageNet dataset has over 400GB of labeled image data. We experimented with two influential models: ResNet[1] and VGG[2]. As mentioned before, DNN training requires time and compute power. One of the most important advantages of using HPC is the ability to use multiple GPUs for training neural networks. Figure 1 represents the simplified schema of the distributed training process.

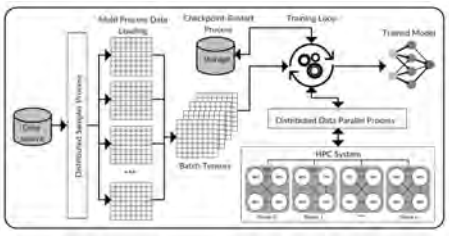


Figure 1: A simplified schema of distributed training

VGG

VGG is a classical convolutional neural network architecture. It is created based on a study of how to increase the depth of convolutional neural networks by Andrew Zisserman in 2014. The network is characterized by its simplicity: the only components being convolutional layers, pooling layers and a fully connected layer.

ResNet

A later more advanced architecture is the ResNet. ResNet is a method developed to avoid the problem of vanishing gradients. Vanishing gradients is a problem encountered when the number of layers is increased causing the gradients in the backpropagation to be infinitesimally small as it moves to the earlier layers, thus making it harder to train. ResNet implements skip connections to gradually increase the depth of the network.

Used Frameworks

During training, we moved in two parallel branches, one of us doing the experiments in PyTorch and the other in Tensorflow. For experiments with distributed training, we used PyTorch data parallel and Horovod, a specialized framework by Uber.

Experiments & Results on Training on HPC

In our experiments we first trained simple models on CIFAR10 and CIFAR100 datasets. Then, we moved on to more exciting experiments with the ImageNet dataset with ResNet and VGG architectures. The most exciting results we got on the training side were the acceleration we got with training on multiple GPUs. We trained a ResNet-152 on ImageNet with 1-2-4-8-16-32 GPUs and got very good acceleration results with the metric of time per epoch.

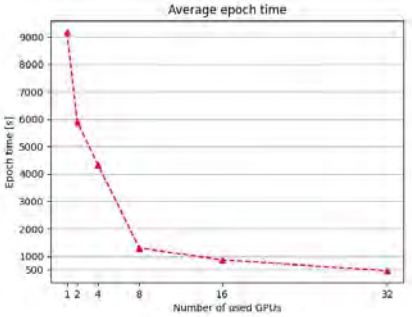


Figure 2: Epoch time for ImageNet training by # of GPUs used

As seen in Figure 2, we got a constant acceleration as the number of GPUs increased. BSC Power9 cluster has 4 GPUs per node. Therefore, after 4 GPUs, we needed to train with multiple nodes. The reason we could not get close to theoretical acceleration after 8 GPUs is the necessity of communication between 3 or 4 nodes.

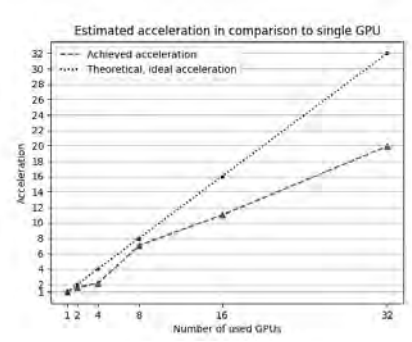


Figure 3: Acceleration in training by # of GPUs used vs. theoretical acceleration

Optimizing ML Models for Edge Devices

What is Edge?

Despite the multiple definitions, edge devices are devices that are close to the end user. There are billions of them and they are not as powerful as HPC servers. Smartphones, Raspberry Pi's, and daily laptops are some of the examples of edge devices.

Why would we move Machine Learning apps to Edge?

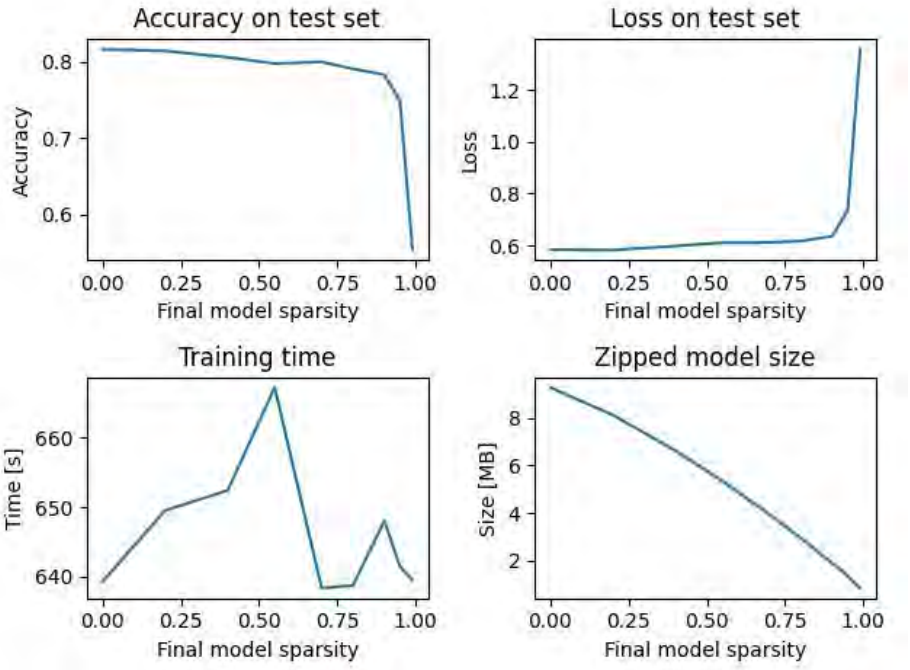
An important question is why would we want to use edge devices instead of deploying to the cloud. Most of the time, deploying to the cloud might become a superior choice, however, there are some important reasons for opting to deploy to edge devices. Firstly, moving to the edge is more secure, as no data is transmitted through the internet. Some specialized applications might have highly sensitive data, therefore choose to go with the edge devices. Secondly, probably the most importantly, ML applications can run offline when deployed to the edge. Thirdly, some systems might require eliminating the latency caused by internet access. Last but not least is that deploying to edge devices is much cheaper than maintaining or renting cloud servers for service providers.

Model Optimization Experiments & Results

There are many techniques to optimize ML models. Few of them are pruning, quantization and knowledge distillation. In our project, we mostly focused on pruning. It is a simple optimization method, but highly effective. With pruning, we can omit the least significant neurons or weights. It can result in reduction of the model size by 5 or more times, and speed up the inference process. We experimented with pruning on relatively shallow models (4-5 layers) and a small dataset (CIFAR 10) to speed up experiments. We compared accuracy and loss on the test set, training time and size of the compressed model.

Challenges of Training ML on HPC

We faced many challenges during our project related to HPC. Therefore, we wanted to share our findings in this report. The first problem we faced was



the big dataset management. ImageNet is a huge dataset and it was not readily available in the cluster. Moving it across the different drives in the cluster was a hindering process. Therefore, we believe that having the data ready prior to starting a project is a very important step. Other than that, cluster maintenance that occurred during our project was also a challenge that lost us a week. Another important challenge was the complexity of distributed computing. We believe that Horovod[3] is a good choice for handling multi node training. PyTorch Data Parallel also provides an easy interface. It is tested on multiple GPUs across a single node and was able to provide theoretical acceleration. However, we were unable to test it with multiple nodes. The last challenge we faced was the long queue times. A good technique we used was asking for very short times when we are still debugging our code. When we were sure, we asked for longer times.

A Test Case: Object Detection with YOLO v3 Tiny on Raspberry Pi

As the last test case of distributed training and optimizing for edge, we worked on implementing an object detection model on a Raspberry Pi 2. First we trained YOLO v3[4] with distributed training on MS Coco dataset. YOLO stands for You Only Look Once and is a state-of-the-art fast model for real-time object detection.

YOLO v3 is then tested on a laptop with GTX 1650 GPU. It was quite successful with identifying objects pretty well at 13 FPS. The model size was 236 MB.

However, this model was not fast enough. Therefore, we optimized the model by using a different version of YOLO v3 called YOLO v3 Tiny and pruning it to get even better results. We trained YOLO v3 Tiny with iterative fine-tuned pruning and in the end our model size was 34 MB.

We used ROS (Robot Operating System) with packages usb_cam[5] and darknet_ros[6] to provide the necessary TCP network for feeding webcam images to our model. We were able to accomplish about 10 FPS with this model as well in a very resource-constrained environment.

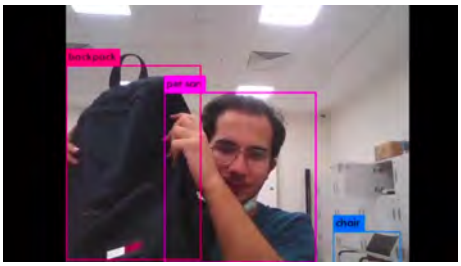


Figure 4: Result of the YOLO v3 Tiny on Raspberry Pi

References

[1] He et al. (2015). Deep Residual Learning for Image Recognition

[2] K. Simonyan and A. Zisserman (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition

[3] A. Sergeev and M.D. Baso (2018). Horovod: fast and easy distributed deep learning in TensorFlow

[4] J. Redmon and A. Farhadi (2018). YOLOv3: An Incremental Improvement

[5] B. Pitzer. USB Camera Package for ROS, http://wiki.ros.org/usb_cam

[6] M. Bjelonic (2009). Darknet package for ROS, http://wiki.ros.org/darknet_ros.

PRACE SoHPCProject Title

Building Resilient Machine Learning Applications(From HPC to Edge)

PRACE SoHPCSite

Barcelona Supercomputing Center, Spain

PRACE SoHPCAuthors

Jakub Raczyński, Mehmet Enes Erciyes

Jakub Raczyński

Mehmet Enes Erciyes

PRACE SoHPCMentor

Leonardo Bautista Gomez, BSC, Spain

PRACE SoHPCContact

Mehmet Enes, Erciyes, Koç University
E-mail: merciyes18@ku.edu.tr

PRACE SoHPCSoftware applied

Python, PyTorch, Tensorflow, Darknet, Horovod

PRACE SoHPCAcknowledgement

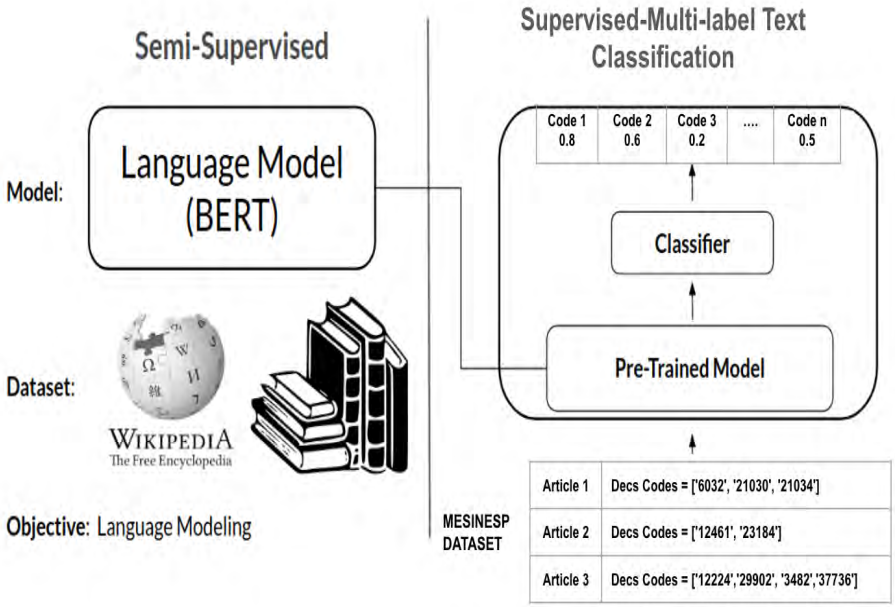
We would like to acknowledge and thank for the support and guidance we got from our mentors, Albert Kahira and Leonardo Bautista Gomez and the kind support team at Barcelona Supercomputing Center.

PRACE SoHPCProject ID

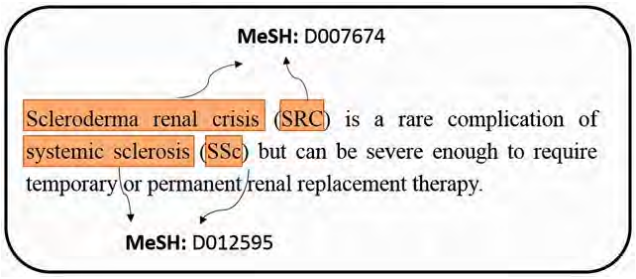
2104

Transfer learning for biomedical texts

Aslihan Uysal



Transfer learning is a technique where a deep learning model trained on a large dataset is used to perform similar tasks on another dataset. We call such a deep learning model a pre-trained model. This project aims to provide a prototype of transfer learning for a classification task that aims to consider the Spanish language as input and compare different pre-trained deep learning models' approaches such as cross-domain and cross-lingual.



An example of extracting mentions from PubMed literature, and assigning MeSH concept identifier for each mention.

Motivation of the Project

Biomedical and Life Sciences are two of the main areas presenting a considerable growth in literature through the last decade, which is demonstrated by the increase in articles indexed in PubMed (a database of biomedical articles).

An example of a BioNLP task that has received increasing attention in the BioASQ challenge that has to be indexed abstracts with multiple labels (i.e., a multilabel text classification), the performance of the proposed systems has increased considerably over the baselines and the current system used by the National Library of Medicine (NLM).

However, this task considers only the abstracts of English articles, not covering other languages with a considerable academic writing volume, such as Portuguese, Spanish, and French.

The goal of this research project is to provide a prototype of classification in the form of a BioASQ submission that may take into account the Spanish language as the input taking advantage of the extreme usage of deep learning architectures of transformers models provided by HuggingFace Library.

The output of the project is planned to be used in:

- Biomedical search engines
- Biomedical question-answering systems

```
{
   "abstractText": "Resumen  Introducción:  Las fístulas carótido cavernosas son malformaciones vasculares infrecuentes que generan un shunt arterio
venoso patológico que compromete el funcionamiento ocular. El diagnóstico definitivo se establece a través de una arteriografía cerebral. Sin embargo
, su carácter invasivo limita su uso en el seguimiento. El objetivo de este trabajo es ilustrar el valor del estudio con ultrasonido doppler transcra
neal para el diagnóstico y describir los parámetros de flujo que pudieran modificarse.  Pacientes: Se realizó una revisión retrospectiva de las histo
rias clínicas de los pacientes atendidos con diagnóstico de fistula carótido cavernosa en la unidad de ictus del Hospital CQ Hermanos Ameijeiras de L
a Habana, entre enero de 2005 y mayo de 2014. Se recogieron variables demográficas y de la enfermedad, así como los resultados de los estudios de ima
gen y ultrasonido.  Resultados: Se describen las características clínicas e imagenológicas de tres enfermos en los que se confirmó el diagnóstico. En
 los dos pacientes con comunicaciones directas, se registró un aumento de la velocidad media de flujo en la vena oftálmica, arterializada, con dismin
ución de la pulsatilidad; sumado a aumento en la velocidad de pico diastólico en la arteria carótida interna ipsilateral a la fístula. En el paciente
 con la fístula indirecta los cambios fueron menos marcados.  Conclusión: El estudio con ultrasonido fue de utilidad en el diagnóstico de las fístula
s carótido cavernosa. Mostró diferencias en parámetros de flujo que pueden servir para clasificar las fistulas.",
   "db": "LILACS",
   "decsCodes": ["12461", "23184", "2576", "28611", "30991", "34246", "34247"],
   "id": "mesinesp-dev-003",
   "journal": "Rev. ecuat. neurol",
   "title": "Fístula Carótido Cavernosa. Utilidad del ultrasonido Doppler en el diagnóstico",
   "year": 2019
}
```


Figure 1: Example of MESINESP Dataset.

The Project Overview

We can consider the project predicting 'DeCS Codes' is as basically a Multi-label Text classification problem.The solution will be based on leveraging the power of the pre-trained Hugging Face Transformers libraries.

Therefore developing a machine learning model that will accurately predict all the indexes that could be associated with the articles will provide us to enrich our purpose of the project which is finding documents as accurately and fastest as we can.

And this brings such an important question that is how to use pre-training models for our task or in other words how to fine-tuning pre-trained models?

How Transformers Models Work?

The out-of-the-box Transformers models have already been pre-trained on such a huge corpus data and thus has a good understanding of generic on a specific language or even multi-lingual texts. However, the particular dataset from MESINESP comprises a lot of medical-related words, which the Transformers models may not have seen during the pre-training phase.

Hence we need to fine-tune the model on our dataset so that it can build an understanding of our dataset and become better at the text classification task. The way to do that is to add a classification head on top of the core Transformers models and then train the entire model on our dataset.

This way the model develops a statistical understanding of the language it has been trained on, but it's not very useful for specific practical tasks. Because of this, the general pretrained model then goes through a process called transfer learning. During this process, the model is fine-tuned in a supervised way — that is, using human-annotated labels — on a given task.

In MESINESP dataset: There are about 370 thousand records for a train set at the Virtual Health Library manually indexed with DeCS codes that are almost 23 thousand unique codes on the entire dataset. We only need "AbstractText" and multi-label structure of the "DeCS Codes" columns as an input for training transformers models.

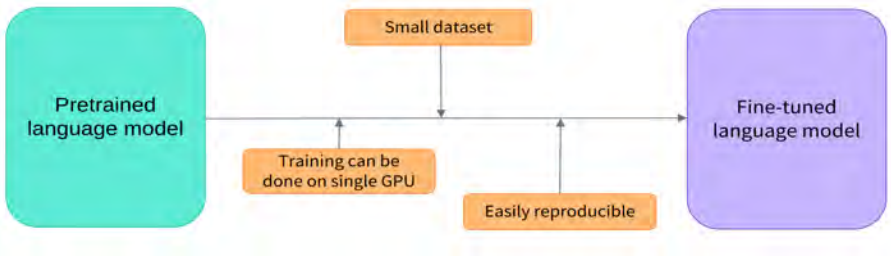


Figure 2: Fine-tuning, on the other hand, is the training done after a model has been pretrained. To perform fine-tuning, you first acquire a pretrained language model, then perform additional training with a dataset specific to your task.

We want to compare the multilingual BERT model (mBERT), the BETO model (Spanish-based BERT) with the in-house RoBERTa biomedical model. Basically, we want to answer the following question:

""To what extent a domain-specific model is better than a general domain model when making a classifier? ""

Load & Preprocess the Dataset

The main libraries we need are:

- Hugging Face Transformers Library (for Models and Tokenizer)
- PyTorch Deep Learning Framework (for Dataset preparation)
- Sklearn (for splitting dataset & metrics)

Many of the DeCS Codes have a very low count. For the scope of this problem, we could restrict ourselves to less than the amount of unique DeCS Codes. That gives us still the same 370 thousand rows of medical texts which are decent enough given that we are using pre-trained models.

Figure 3: Frequency of DeCS Codes.

The histogram plot reveals that there are almost 10 thousand unique DeCS codes that occurred in less than 10 in the entire dataset.Also in general, reducing that many unique codes still could be reasonable for the model to be able to perform classification.

Training Phase

We need to tokenize and encode the text data numerically in a structured format required for all transformers models, the BERT Tokenizer class from the Hugging Face library makes this a simple affair.

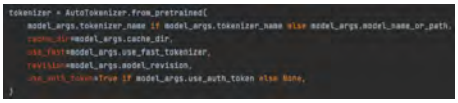


Figure 4: Huggingface BERTTokenizer Example

Here, we can see extracted values of a tensor of an example sentence.

```
output = tokenizer.encode("Hello, y'all! How are you 😁 ?")
```

Figure 5: A Sample Sentence for Encoding

```
print(output.tokens)
# ["Hello", ",", "y", "'", "all", "!", "How", "are", "you", "[UNK]", "?"]
```

Figure 6: Output Tokens of Sample Sentence

```
print(output.ids)
# [27253, 16, 93, 11, 5097, 5, 7961, 5112, 6218, 0, 35]
```

Figure 7: Output Ids of Sample Sentence

In MESINESP Dataset every "abstractText" converted into a numerical format which has attributes.And similarly, every "decsCodes" feature is converted into a list format which is unique Decs Codes length and takes values 1 if that particular example has that code otherwise takes 0.

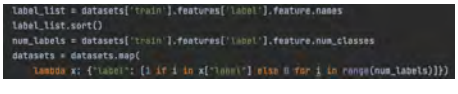


Figure 8: DeCS Codes Representation for Model Input

Updating weights process is based on computed total loss for every batch after every iteration model weights can be modified for getting nearest score for our ground truth which is the real label.

Since the output is multi-label (multiple tags associated with an article), we may tend to use a Sigmoid activation function for the final output and a Binary Cross-Entropy loss function. However, the Pytorch documentation recommends using the BCEWithLogitsLoss() function which combines a Sigmoid layer and the BCELoss in one single class instead of having a plain Sigmoid followed by a BCELoss.

We can see that it combines a Sigmoid layer and the BCELoss in one single class.

$$\ell_c(x,y) = L_c = \{l_{1,c},\dots,l_{N,c}\}^\top, \quad l_{n,c} = -w_{n,c}\left[p_c y_{n,c}\cdot\log\sigma(x_{n,c}) + (1-y_{n,c})\cdot\log(1-\sigma(x_{n,c}))\right]$$

where c is the class number ($c > 1$ for multi-label binary classification, $c = 1$ for single-label binary classification), n is the number of the sample in the batch and p_c is the weight of the positive answer for the class c

Figure 9: BCEWITHLOGITSLOSS Function

Benchmarking & Evaluation

We have already labeled the dataset as a test set indexed manually by seven experienced medical literature indexers. And we will use this dataset for benchmarking.And Using the best performing trained model we can start predicting DeCS Codes that can be associated with any relevant abstractText that we have.

To be able to measure the performance of the model;

- We can use **Accuracy** metric as the main control metric

Table 1: Accuracy Results Table for Epoch 1

Model Name	Epoch	Accuracy
mBERT	1	0.873
BETO	1	0.8723
BioRoBERTa	1	0.8704

Table 2: Accuracy Results Table for Epoch 4

Model Name	Epoch	Accuracy
mBERT	4	0.8596
BETO	4	0.8613
BioRoBERTa	4	0.8716

According to final results, each models have very close accuracy to each other. Increasing epoch from 1 to 4 does not improve accuracy much as the models might started to overfit the training dataset more thus accuracy might have started to reduce at some point. To overcome this obstacle, EarlyStoppingCallback class from Huggingface library can be used for future improvements to be able to prevent and stop learning once models might reduce the accuracy.

Conclusion

In this article, I have described a Multi-label Text Classification comparing different 3 pre-trained Transformers models' performances on such big data and with a huge number of labels.

For getting better results improving accuracy as future improvements:

- HyperParameter Search technique could help us to be able to get the right parameters for later training processes such as number of epochs or learning rate.
- To be able to reduce the size of label dimension, we can also consider some clustering techniques grouping most similar labels into one label.

References

[1] HuggingFace Transformers Models Official Tutorial

[2] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

PRACE SoHPCProject Title

Cross-Lingual Transfer learning for biomedical texts

PRACE SoHPCSite

Barcelona Supercomputing Centre, Spain

Aslihan Uysal

PRACE SoHPCAuthors

Aslihan Uysal, [ITU,] Turkey

PRACE SoHPCMentor

Marta Villegasšič, BSC, Spain
Casimiro Carrinošič, BCS, Spain

PRACE SoHPCContact

Aslihan, Uysal, ITU
Phone: +905398377980
E-mail: uysalas20@itu.edu.tr

PRACE SoHPCSoftware applied

Python, Pytorch
https://github.com/aslihanuysall/SoHPC_2021

PRACE SoHPCAcknowledgement

Special thanks to the Head of Text Mining Unit at BSC Marta Villegas & Research Engineer at BSC Casimiro Carrino for their support and guidance throughout the whole project.

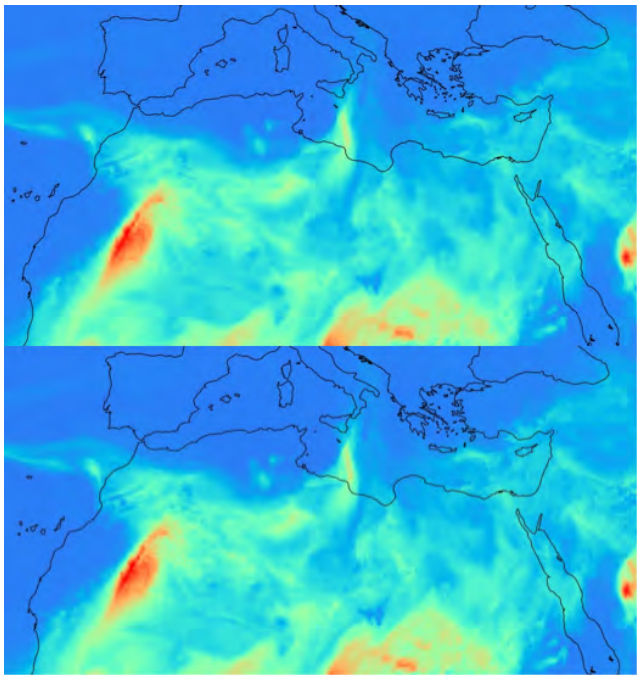
PRACE SoHPCProject ID

2105

Improvement of a Python package providing multiple standardized interpolation methods for atmospheric chemistry models

Interpolation Tool

Brian O'Sullivan, Daniel Cortild



Measurements and predictions are not always acquired in the wanted or needed form. The Interpolation Tool developed at BSC transforms raw data into different types of grids.

Interpolation is a mathematical principle used to determine data points where no measurement has been taken. Using one of the various interpolation methods available (weighted average methods, regression models, etc.), one can estimate, i.e. interpolate, a value at a previously unmeasured point to a high accuracy. Interpolation is typically a general mathematical model, and can therefore be computed in any domain, such as across time, or across three dimensional space. At BSC, an interpolation tool has been created which can be used on output data from a variety of atmospheric models. The tool proves extremely useful for changing the shape of the output grid (i.e. changing from one grid to another, or from one grid to a set of points) and can even improve the resolution of a dataset. The title image above is an example of this The first figure (above) is experiment a2in, an atmospheric dust report over Southern Europe and Northern Africa produced by the NMMB/MONARCH model from BSC. The second image (lower) is this same model interpolated on to a different standard grid with a higher resolution.

For the interpolation tool, interpolation is carried out across two domains. First, there is horizontal interpolation where a geographical grid is interpolated in two-dimensions (longitude and latitude). Secondly, vertical interpolation is also an option. For vertical interpolation, the points at which values are calculated lie between two different vertical levels within the atmosphere. In other words, the spatial dimension interpolated across is altitude.

Spatial Interpolation

For horizontal interpolation, a weighted average method known as inverse distance weighting is used. Fig. 1 demonstrates the structure of a weighted average method, where we wish to calculate some value at the central point (for example, temperature). This is done by using the surrounding points to evaluate a weighted average

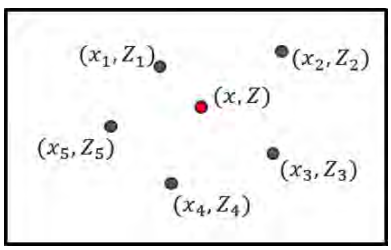


Figure 1: Spatial interpolation example.

By using measured values at known points within the vicinity, we can formulate an expression. This is done by assigning a point weighting to each nearby point, represented by λ_i. A point weighting determines how much of a contribution each known point makes to our final estimation, where points with a higher weighting make a greater contribution. Such weights are usually given depending on the distance to the unmeasured point. The following equation demonstrates this, where we sum up over every known point according to their value and point weighting to compute a final interpolated value.

Vertical Interpolation

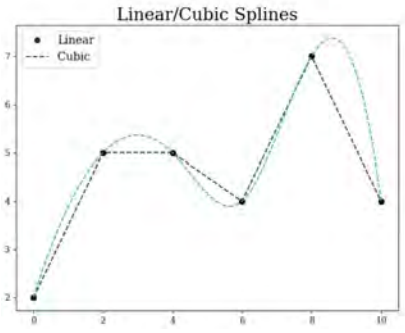


Figure 2: Vertical interpolation example.

Unlike horizontal interpolation, vertical interpolation is performed using a cubic spline method. The data on missing levels is interpolated according to the data from known vertical levels using the same horizontal coordinates. For the cubic spline method, a cubic polynomial is fit between levels in a piece-wise manner.

Our Project

Our project was to improve this pre-existing Interpolation Tool developed at BSC and ran on the Nord3 machine. Originally, the tool could only run as a standalone tool, being executed from the command line using a static configuration file. This means that all the data must be stored in files which can be referenced in a configuration file. The first task consisted of allowing the tool to be run from a Python script, using data already loaded into memory.

The second task was to improve the range of the interpolation results of the tool. Before this summer, the tool could only interpolate from one standard regular grid to another standard regular grid. However, in some situations it is preferable to interpolate to a given set of points instead of an entire grid. Given how the tool was implemented for horizontal interpolation (Using the weighted average method), this could be solved simply by tweaking the way the weight matrix was computed.
Our third task was to reverse the order of horizontal and vertical interpolation in the tool.

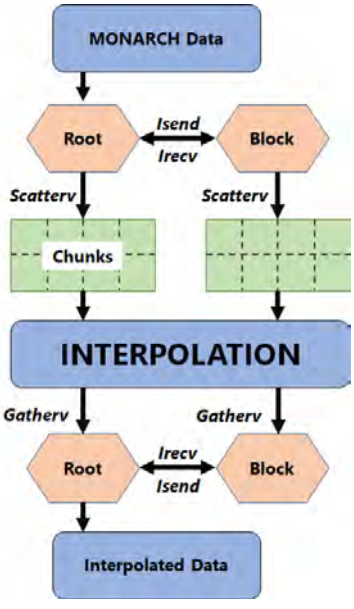


Figure 3: Reversing Interpolation Order.

Currently, horizontal interpolation is executed first and is serialised, whereas vertical interpolation is parallelised using mpi4py. The dataset is split into chunks before interpolation, grealty improving computation time. By reversing the order, we could parallelise horizontal interpolation, further improving performance. Figure 3 demonstrates the tool's current state, the change is shown in Figure 4.

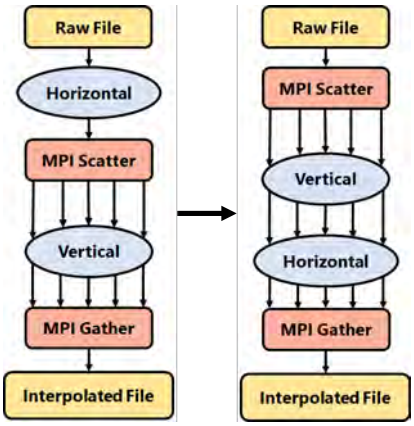


Figure 4: Reversing Interpolation Order.

Unfortunately, horizontal interpolation relies on a file generated at the start of the job using the entire dataset. Therefore, indexing problems occurred between data shape and the weight matrix file. Normally a mask of the original data shape would be a simple solution, containing the data chunks at their original indexes. However, this process leads to the Nord3 killing the job, as the current numpy mask implemented requires far more data than the original chunks (30x bytes for the a2in experiment). Structurally, this change should be close to completion, but this memory bug will need to be solved before any further evaluation can be done.

Another task we were working on was solving a deadlock caused if both horizontal and vertical interpolation flags were active at the same time. This bug was strange since running them separately did not cause any problems, but they could not run together. This suggested some kind of incompatibility between libraries, which was indeed the case. Mpi4py, the MPI package for Python, and the multiprocessing libraries were causing a conflict. This occurred whenever an MPI scattering was ran between two multiprocessing pools. This situation would only occur once both flags were active, ultimately causing the deadlock. We managed to find an unfortunate and unsatisfactory solution which was to serialize this section of the code.

Results

We have produced an improved version of the interpolation tool. The tool can now be called using a standard Python script, and the option of interpolating to a set of points has been added. The order of interpolation of the tool has been reversed, enabling the parallelisation of horizontal interpolation. Finally, the freezing bug has been fixed, allowing both horizontal and vertical interpolation in the same job. Overall, these changes allow greater flexibility in the tool's usage for any future analysis.

Although much progress has been done on the interpolation tool, there is much more potential to further increase the tools flexibility and scaleability. Examples include enabling interpolation onto other unique grids, such as a rotated grid or GCC grid. A current technique known as pickling is used to free up memory, but could possibly be replaced by a more optimal technique that would optimize horizontal interpolation overall.

PRACE SoHPCProject Title

Improvement of a python package to provide multiple standardized interpolation methods for atmospheric chemistry models

PRACE SoHPCSite

Barcelona Supercomputing Center, Barcelona, Spain


Brian O'Sullivan


Daniel Cortild

PRACE SoHPCAuthors

Brian O'Sullivan, Daniel Cortild

PRACE SoHPCMentor

Francesco Benincasa, BSC, Barcelona, Spain

PRACE SoHPCContact

Name, Surname, Institution
Phone: +12 324 4445 5556
E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied

Virtuoso

PRACE SoHPCMore Information

www.virtouso.org

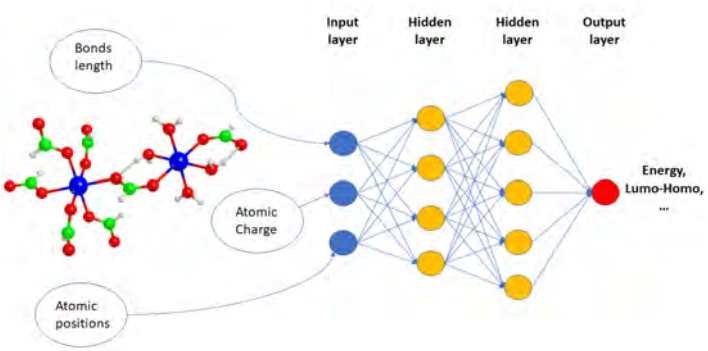
PRACE SoHPCAcknowledgement

Thank you to our mentors Francesco Benincasa and Kim Serradell. Also thank you to all at PRACE and BSC.

PRACE SoHPCProject ID

2106

Neural Networks in Quantum Chemistry



Joseph Sleiman & Scott le Roux

The current standard for solving quantum chemical problems scales badly with increasing molecular size. Thus, Neural Networks are proposed as an alternative solution to determine the structure-property relationships of molecules accurately and more efficiently. Using *TensorFlow* and the *DScribe* molecular descriptor library, we implemented Neural Networks that predicted the internal energies of organic molecules in the QM9 dataset, achieving a mean squared error of 0.002 Hartree^2. During descriptor creation and Neural Network training, we found that GPUs achieved significant speedups over single CPU computations, but were slower than parallel CPU computations using 8 cores or more.

Machine Learning (ML) and Artificial Intelligence (AI) have provided comprehensive solutions to various natural and scientific phenomena, ranging from galaxy detection to genome sequencing, and paired with the boom in data and growing computational power, means the applications for ML and AI will continue to stretch further and wider into even the most established and niche sectors.

Likewise, quantum chemistry - a branch of chemistry focused on predicting chemical and physical properties of molecules and materials[1] - has seen an increase in ML applications, particularly through the use of Neural Networks (NNs). This Deep learning (DL) architecture consists of groups of "neurons" (a term inspired by the architecture and functionality of the brain) all densely connected to form a network where the input is typically a vector consisting of your data representation and the output is a value representing your prediction. In between these layers, there are hidden layers consisting of an arbitrary number of neurons optimised by training weights to produce the best predictions possible on the training datasets, whilst also generalising to the new data we wish to make predictions about.

Accordingly, in this project we use NNs to predict quantum mechanical properties of molecules, instead of the traditional method of Density functional Theory (DFT), which uses approximations to the Schrödinger equation. This is a very promising avenue considering that it gets increasingly difficult and computationally expensive to solve the Schrödinger equation for molecules with dozens of atoms or more. Hence, the numerical optimisation approach inherent in ML could result in accurate predictions without the need for large computing power and a lot of time! This is possible because NNs can approximate any function when combined with multiple hidden layers and neurons with nonlinear activation functions.

How Did We Replace the Schrödinger Equation?

The main issue with using ML to solve quantum chemical problems is that

chemical data does not come in a desirable form readable by NNs. To bridge this, we used "molecular descriptors" which transform raw chemical data into feature vectors which preserve specific chemical properties. These descriptors were implemented via the Python library DScribe[2] which provides molecular descriptors for use in ML applications. More specifically, we used two global descriptors - the Coulomb Matrix (CM) and Many-Body Tensor Representation (MBTR) - and two local descriptors - Smooth Overlap of Atomic Positions (SOAP) and Atom-Centred Symmetry Functions (ACSF) - throughout our project in order to output property predictions such as the internal energy of a molecule at zero Kelvin (0K), and the charges on each atom making up the molecule.

For example, the CM is an $N \times N$ matrix where N is the number of atoms in the molecule. The diagonal elements represent the self-interactions of the i^{th} atom, and the off-diagonal entries represent the coulombic repulsion between the i^{th} and j^{th} nuclei. This produces a compact matrix (see Figure 1) that encodes the positions and electrostatic interactions between all the atoms in the molecule (refer to this link for more details on all the descriptors used).

In order to attain sufficiently accurate NN models, we require a lot of data during training. Hence, we utilised the QM9 dataset[3] which contains 134,000 organic molecules with their corresponding energetic and thermodynamic properties calculated very thoroughly and laboriously via traditional quantum chemical analytical methods. This dataset is readily accessible online and is thus used as a benchmark for developing new quantum chemical tools, as well as for comparative purposes when publishing new research. In terms of data handling, we sorted the QM9 dataset in increasing value of internal energy at 0K, and then reserved every fifth data point for the test set, ensuring that the test was representative of a range of internal energies. The remaining data was randomly separated into an 80:20 splitting of training and validation sets, respectively. The end goal of the project is a supervised learning task, in which we train the algorithm on a labelled dataset (QM9) in order to predict a final output label (internal energy at 0K); this is the standard regression or classification ML procedure.

We implemented our NNs via the DL Python library *TensorFlow* with *Keras* 2.0 back-end, and used the software package KerasTuner to optimise the hyperparameters of the network in order to minimise the mean squared error (MSE) of our predicted results versus the true values given by QM9. These hyperparameters included the learning rate, the batch size, activation function, weight initialisation, as well as the number of neurons and hidden layers making up the network topology. This approach uses either an iterative random-search optimisation or a Bayesian optimisation that minimises your desired loss function (MSE, in our case) in a systematic manner, as opposed to a very long and laborious brute-force method of optimising the NN hyperparameters. This task was completed separately for each molecular descriptor because there is no universally optimal NN architecture for any given problem (see "No Free Lunch"[4] theorem), and each descriptor has a different number of features (see Table 1), altering the number of neurons needed in the input layer.

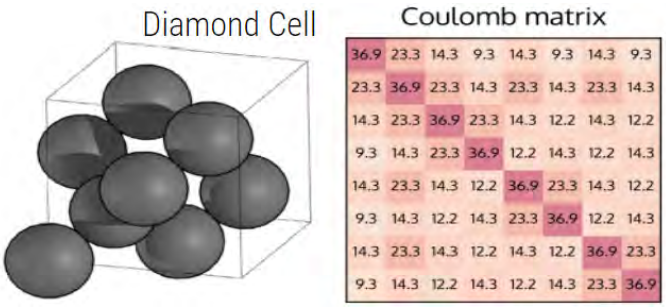


Figure 1: Conversion of the positional and electrostatic information of a diamond cell into its Coulomb Matrix form via *DScribe*.[2]

Finally, for the High-Performance Computing (HPC) part of the project, we benchmarked the speeds of both descriptor creation and NN training using GPUs and CPUs, parallelising the latter with multiple cores. This was achieved using the remote server computing node provided by our host site at the Slovak Academy of Sciences, where we were given access to two Intel Xeon CPUs and an NVIDIA Tesla K20m GPU.

Neural Network Model Results and Benchmarking

Optimization of NNs is the heavy lifting of the project; in order to save a lot of time, it is imperative that educated guesses are made concerning what hyperparameter options to test before running the tuning via *KerasTuner*. Batch size and learning rate took the accepted standard values of $32, 64, 128$ and $.01, .001, .0001$, respectively. For weight initialisation, we tested *He Uniform* and *Xavier Uniform* initialisers, which are considered the state-of-the-art and either is compatible with the majority of activation functions. The activation functions we tested were the *rectified Linear Unit* (ReLU), *exponential Linear Unit* (ELU), *softplus*, and *shifted softplus* (ssp),[5] described by the following equations:

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

$$ELU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ e^x - 1, & \text{if } x < 0 \end{cases}$$

$$softplus(x) = \ln(e^x + 1)$$

$$ssp(x) = \ln(0.5e^x + 0.5)$$

Finally, we tested different NN topologies selecting from between 1-4 hidden layers each with a variable number of hidden neurons from the set, $\{32, 64, 128, 256\}$.

In Table 1, we see the optimised NN structures for each molecular descriptor and their corresponding predictive accuracy. While the best hyperparameters varied across descriptors, a batch size of 32 and an *ADAM* optimiser were universally optimal. To summarise, the CM achieves the best MSE by an order of magnitude over the second best descriptor, MBTR. The latter was a further order of magnitude more accurate than the local descriptors ACSF and SOAP.

In terms of descriptor creation, as we can see from Figure 2, the speed up ratio, $R = \frac{t_{serial}}{t_{parallel}}$, for complex descriptors with thousands of features (ACSF, SOAP, MBTR) consistently exceeds 2 and reached a maximum of $\sim$10 for 16 CPU cores on MBTR. This highlights the importance of parallelising the creation of descriptors with 3000 features or more. Conversely, in the case of simpler descriptors like the CM, invoking parallel implementation produces an overhead greater than the eventual speed up achieved using multiple cores. Hence

Table 1: Summary of Optimised Molecular Descriptors and their Internal Energy Prediction Mean Squared Error (MSE)

Descriptor	# Features	Learning Rate	Activation Function	Hidden Layer Topology	Weight Initialiser	MSE
CM	841	0.0001	*softplus*	[256]	*Xavier Uniform*	0.002
MBTR	9500	0.001	*ELU*	[64,64,32,128]	*Xavier Uniform*	0.01
SOAP	4920	0.001	*ELU*	[256,32,128]	*He Uniform*	0.241
ACSF	3190	0.0001	*softplus*	[128,32,16]	*He Uniform*	0.521

with R < 1, it is not efficient to parallelise in this scenario.

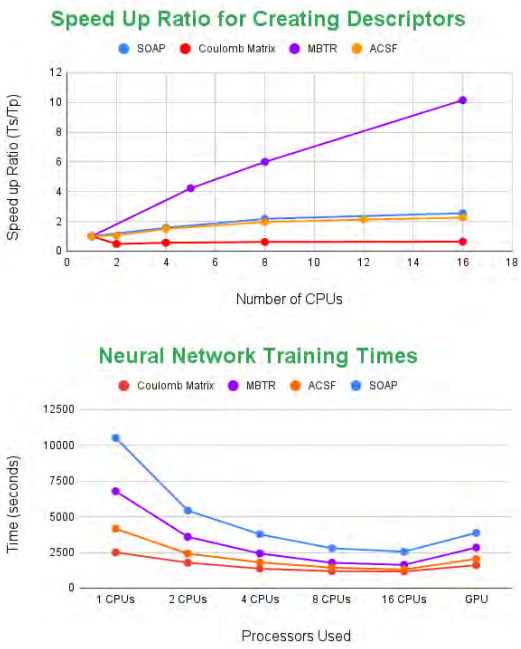


Figure 2: Graphs for computation times

For NN training, we found that there is an inverse relationship between time and number of CPU cores used in the training process, as seen in Figure 2. This is an expected result and highlights that one should aim to utilise multiple cores in order to achieve the best performance. It is also clear that a single GPU core outperforms a single CPU core by $\sim$ 2x. However, if we use 8 CPU cores or more, the training process is consistently faster than a single GPU.

Final Thoughts

Ultimately, the CM was the best molecular descriptor in capturing the salient features of molecules. This is a surprising yet satisfying result when one considers that the CM is the simplest descriptor with the fewest features by a significant margin (see Table 1) and is therefore the most computationally efficient. It was also expected that descriptors that focus on global features like the CM and MBTR would produce the best predictions on global quantities like the internal energy of a molecule at 0K and this proved to be the case.

On the HPC front, as expected, GPUs provided significant speedup to single CPU computation. Reducing computation time during NN training is extremely important and the improvements in hardware such as GPUs and more recently Tensor Processing Units (TPUs) are the primary reason that DL is a leading method in modern computing. These improvements have allowed researchers and programmers to effectively use large DL models to solve problems which previously took many years, in only a few hours.

However, contrary to general computing consensus, we found that parallelised CPU cores were faster than even the GPU speeds. We speculate that this unexpected result was either due to a malfunction with the NVIDIA GPU used, or perhaps *TensorFlow* has optimized GPU calculations for newer GPU hardware. In future work, we hope to solve this GPU issue and validate the widely accepted fact, that GPUs are superior execution platform for NNs to CPUs.

Additionally, we hope to use NNs with a multi-neuron output layer in order to predict other molecular properties such as the charge or spin of each atom that makes up the molecule and we postulate that descriptors like ACSF, focusing on local molecular features, will produce the best results.

References

[1] T. A. Profitt and J. K. Pearson, "A shared-weight neural network architecture for predicting molecular properties," *Phys. Chem. Chem. Phys.*, vol. 21, pp. 26175–26183, 2019.

[2] L. Himanen, M. O. Jäger, E. V. Morooka, F. Federici Canova, Y. S. Ranawat, D. Z. Gao, P. Rinke, and A. S. Foster, "Dscribe: Library of descriptors for machine learning in materials science," *Computer Physics Communications*, vol. 247, p. 106949, 2020.

[3] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific Data*, vol. 1, 2014.

[4] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[5] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter convolutional neural network for modeling quantum interactions," 2017.

PRACE SoHPC Project Title

Neural Networks in Quantum Chemistry

PRACE SoHPC Site

Slovak University of Technology, Bratislava, Slovakia

PRACE SoHPC Authors

Scott le Roux, Trinity College Dublin, Ireland
Joseph Sleiman, University of Edinburgh, UK

PRACE SoHPC Mentor

Marian Gall, Slovak University of Technology, Bratislava, Slovakia

PRACE SoHPC Contact

Scott le Roux, Trinity College Dublin
E-mail: sleroux@tcd.ie
LinkedIn: scottleroux
Joseph Sleiman, University of Edinburgh
E-mail: s1766037@sms.ed.ac.uk
LinkedIn: josephsleiman


Scott le Roux


Joseph Sleiman

PRACE SoHPC Software Applied

Python, TensorFlow, DScribe, Keras-Tuner

PRACE SoHPC More Information

Joseph's PRACE GitHub Repository
Scott's PRACE GitHub Repository

PRACE SoHPC Acknowledgment

Special thanks to Marián Gall for his support and guidance during this project, to everyone at PRACE for the opportunity, and the team at ICHEC for their insightful training week programme.

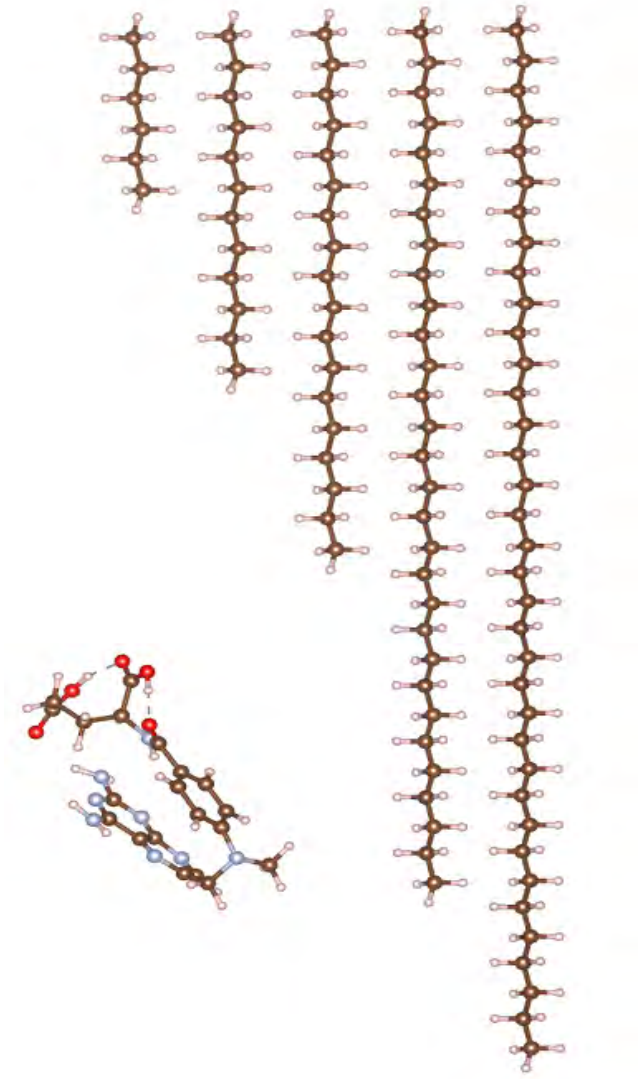
PRACE SoHPC Project ID

2107

Efficient Fock matrix construction in localized Hartree-Fock method

Speeding up the HF method

Ioannis Savvidis, Zsurka Eduárd



The scope of our project is to improve an already working algorithm that calculates the ground state solution of large molecules. The algorithm is an implementation of the Hartree-Fock method. By using the Sparse BLAS library, which is specifically designed to handle the multiplication of sparse matrices, we are planning to speed up the algorithm.

The Hartree-Fock (HF) method is used in computational physics and chemistry to determine an approximation of the wave function and the energy of molecules. The algorithm starts with an initial electron density of the molecule, which is inserted in the Hamiltonian,[1] and it solves the time-independent Schrödinger equation. After the first run, the algorithm provides a new guess for the electron density. Next, this guess is plugged back into the Hamiltonian and the Schrödinger equation is solved again. The result will be slightly different, due to the electron density being different from the initial guess. This cycle is repeated until the electron density doesn't change anymore. This cyclic structure is the reason why the HF it is also called the self-consistent field method. A graphic representation of the algorithm is presented in Fig. 1.

In reality, it's not the Hamiltonian that is used in the calculation, but another matrix that already contains the Hamiltonian of the molecule, which is called the Fock matrix.[2] The implementation of the HF method by Noga and Simunek[3] replaces the usual step of diagonalizing the Fock matrix (see Fig. 1), with a simple matrix-matrix multiplication. If we are considering the parallelization of the algorithm, the diagonalization is an unwanted step.[3] As such, this implementation of the HF method can be easily parallelized, since the formulation of the algorithm is diagonalization free.

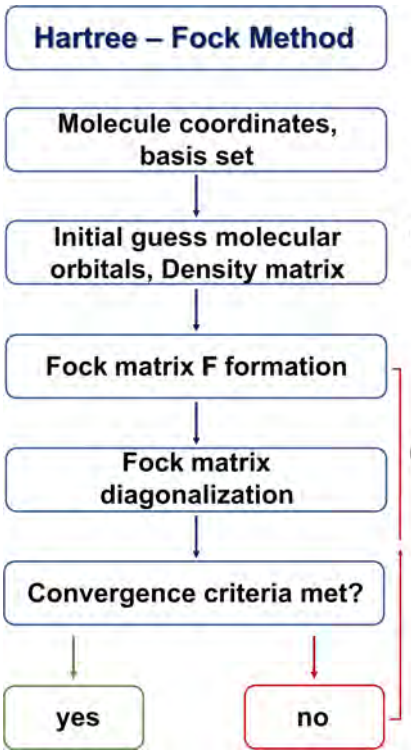


Figure 1: Schematic of the Hartree-Fock algorithm.

In the matrix-matrix multiplication that replaces the diagonalization step, one of the matrices is the electron density matrix. For large molecules containing thousands of atoms, each atomic orbital will overlap only with the neighboring orbitals, leading to only a few non-zero elements in the density matrix. Such matrices, that have only a few non-zero elements, (at most half of the matrix) are called sparse matrices (as opposed to dense matrices). When working with sparse matrices, it is beneficial to use specialized functions and data structures that take advantage of

the sparse structure of the matrix. This can lead to an improvement not only in the memory use of the code, but also in the execution time.

The main goal of our project is to modify the existing algorithm, by taking advantage of the sparsity of the density matrix. We used a sparse matrix–dense matrix multiplication function tailored specifically for sparse matrices and looked at the ways it improves the performance of the algorithm. We also studied how the allocation of the computing nodes and cores of the supercomputer can be used in speeding up the calculations. Our results show that for small molecules the new algorithm is slower, but for alkanes containing 24 carbon atoms or more, on average there is a slight improvement in the run time. We also found that the run time of the code doesn't necessarily increase with the number of cores, and that the number of nodes used doesn't decrease the run time significantly.

The matrix–matrix multiplication

The implementation of the algorithm outlined in[3] is written in Fortran. In order to have access to the sparse matrix–dense matrix multiplication function, we searched for an implementation in Fortran of the Sparse BLAS library,[4] short for (Sparse Basic Linear Algebra Subprograms). The Sparse BLAS library is a collection of functions and data structures fit for handling sparse matrices. Since these libraries are on average 20 years old, we hit multiple dead-ends when trying out different implementations. Eventually we found a working implementation of Sparse BLAS on a StackOverflow post.[5]

The matrix–matrix multiplication we were tasked to replace takes the form

$$M = N \cdot D, \tag{1}$$

where M, N are some matrices and D represents the density matrix. The multiplication function that the Sparse BLAS library provides, requires that the first matrix in the multiplication be sparse. Thus, we transpose eq. (1) and insert

$$M^T = D \cdot N^T \tag{2}$$

into the function we use, where we use the property of the density matrix that $D = D^T$. We thus add two additional steps to the algorithm in the form of transposing matrix N, and having to transpose back M^T. Another complication that arises is that the data structures used in the original code and the Sparse BLAS functions are different. In the original code, the matrices are stored in one-dimensional arrays, while the Sparse BLAS function uses two-dimensional arrays. Unfortunately, all of these additional steps end up slowing down the calculation.

Improvement in the run time

To compare the performance of the new and the original codes, we used multiple molecules: methotrexate ($C_{20}H_{22}N_8O_5$) and 5 alkanes C_6H_{14}, $C_{12}H_{26}$, $C_{18}H_{38}$, $C_{24}H_{50}$, $C_{30}H_{62}$, $C_{36}H_{74}$. A visual representation of these molecules can be seen in the top right corner of the first page. The 5 chains of increasing length are the alkanes and the molecule below is methotrexate.

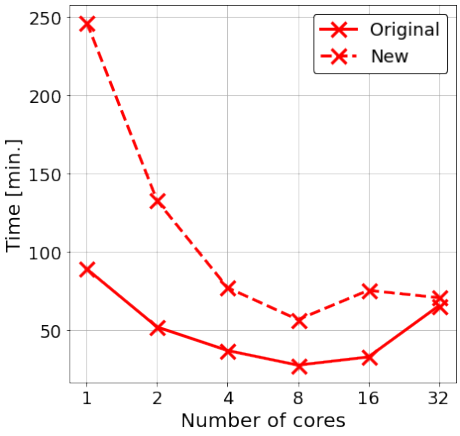


Figure 2: Runtime of the original and new code for methotrexate. The original code is faster for any number of cores.

We used a single computational node and started calculations on 1, 2, 4, 8, 16 and 32 of the cores within the node. Firstly, we studied the run time of the codes for methotrexate. The results can be seen in Fig. 2.

It seems that in the case methotrexate, the new algorithm is definitely slower, no matter the number of cores we use. Since we expect that the sparsity of the matrix will become more important the larger the molecule, we have chosen to study other larger molecules. The size of the density matrix is given by the number of basis functions and in the case of methotrexate, we use 572 basis functions. From the 5 alkanes $C_{30}H_{62}$ and $C_{36}H_{74}$ have considerably more basis functions, 730 and 874, respectively. We calculated the relative improvement in run time for $C_{30}H_{62}$ and $C_{36}H_{74}$, given by

$$\eta = \frac{T_{new} - T_{original}}{T_{original}}, \tag{3}$$

where T_{new} and $T_{original}$ are the run times of the new and original code, respectively. The calculation were performed for a multiple numbers of cores, similarly to the previous case. The results are shown in Fig. 3.

We can clearly see, that there's a slight improvement in the run time of the new code, except for the case of 32 cores. We can get a better idea of the improvement in run time, by calculating the average improvement in run time for each alkane. We show our results in Fig. 4a, where we excluded the 32 core case. We can see that for the case of $C_{36}H_{74}$ a slight improvement of $\approx 4.2\%$ in the run time on average is present.

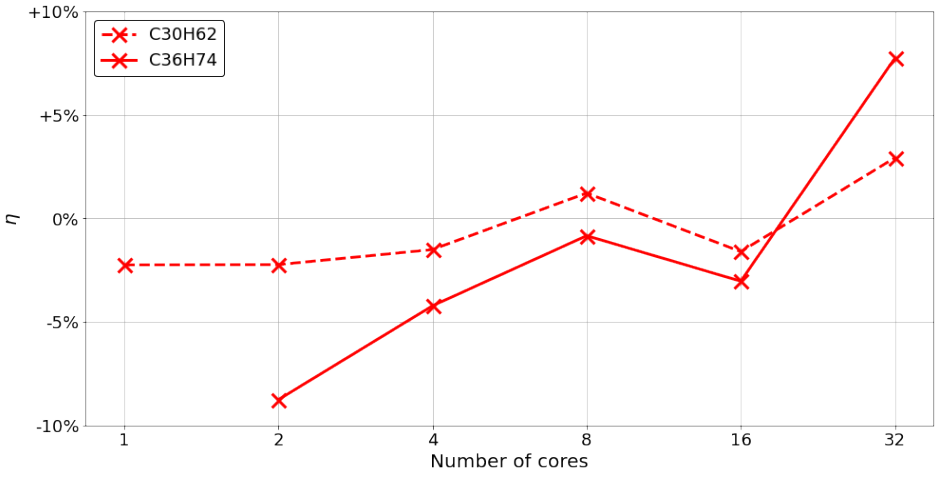
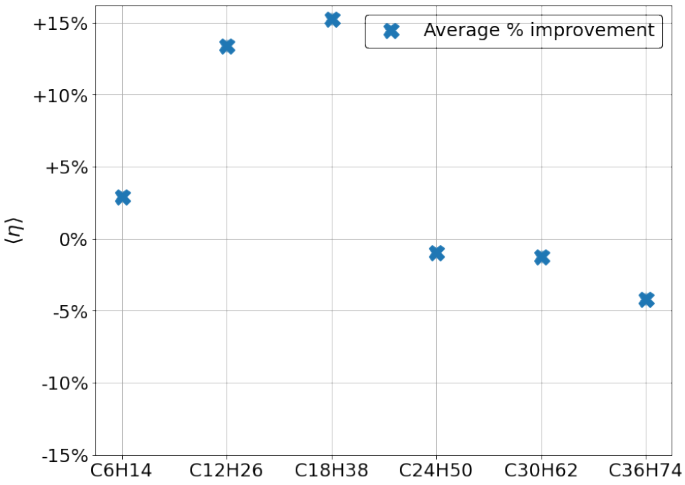
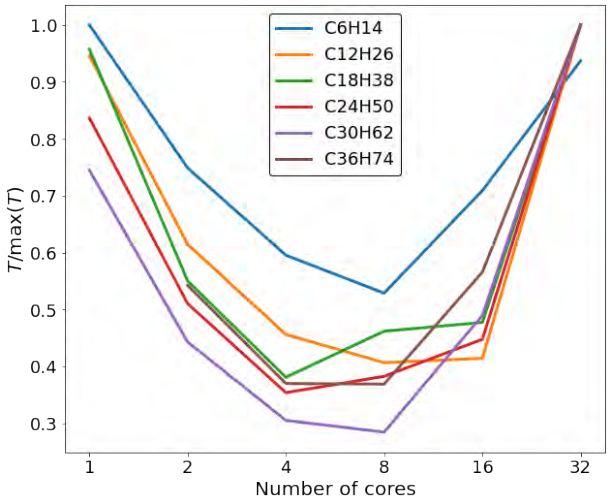


Figure 3: Improvement in the run time of the code given in eq. (3), for $C_{30}H_{62}$ and $C_{36}H_{74}$. Aside the 32 core case, the new code is faster for $C_{36}H_{74}$.



(a) The average improvement in run time of the new code, given by eq. (3), for each alkane.



(b) The run time of the new code, for different number of cores, calculated for the 5 studied alkanes.

Figure 4:

In order to have a clearer picture of the improvement coming from the modified matrix–matrix multiplication, we would either have to study the molecules using a different set of basis functions, or choose larger molecules. Unfortunately, we encountered some numerical problems when trying a number of different basis functions, and we couldn't verify whether these results can be extrapolated for larger molecules or basis function sets. Nonetheless, this is a step that could further the results of this study.

The number of cores

As we have seen before in the case of methotrexate, the run time of the code doesn't always decrease with the number of cores employed in the calculation. In Fig. 2 we can observe that both versions of the algorithm are get faster as we increase the number of cores from 1 to 8 cores, but at 16 and 32 cores they slow down. This can be attributed to the structure of the nodes. We use 4 IBM Power 7 nodes, each with 32 cores and 3.61 GHz of a supercomputer located in Žilina, Slovakia. Each node is equipped with 4 CPUS. Therefore, in the 16 core case, we can attribute the increase in the run time to the two CPUS trying to slow down the calculation, instead of speeding it up. The same behavior can be observed for the alkanes we studied. We choose the same numbers of cores as previously, and we present the run time of the new code divided by the maximal run time for a given alkane. The results are presented in Fig. 4b.

One can clearly see a minimum in the run times at 8 cores (except for $C_{18}H_{38}$). A poor choice in the number of cores can even lead to a threefold increase in the run time (1 and 32 cores). It's also worth mentioning, that in the case of the large molecules $C_{30}H_{62}$ and $C_{36}H_{74}$ we obtain a good run time with 4 cores also. The original implementation of the algorithm yielded similar results.

We also studied how the number of nodes used in the calculation affects the run time. We used 8 cores per node and we increased the number of nodes form 1 to 4. We observed a non-linear decrease in the run time and surprisingly, when we used 4 nodes instead of 1, we only saw a 7% decrease in the run time. We therefore concluded, that it's sufficient to use only 1 node in our calculations.

Discussion & Conclusion

In this work, we have showed that by using a matrix multiplication function specifically written to handle the sparse structure of the electron density matrix, one can improve the performance of the Hartree-Fock algorithm. Although we were limited by time, we have shown that there are strong signs that the modified code can help speed up the calculations for large enough molecules, such as the average decrease in run time of $\langle\eta\rangle = 4.2\%$ in the case of $C_{36}H_{74}$ (see Fig. 4a). In order to confirm our findings, further research is need, using larger molecules or larger set of basis functions. Finally, we also studied how the number of cores and nodes used during the calculations can affect the performance of the algorithm.

Acknowledgements

We would like to thank Ján Šimunek for helping us to better understand the Hartree-Fock algorithm, and showing us how the theory is applied in practice. We are deeply thankful for his guidance during our project.

References

[1] In quantum mechanics, the Hamiltonian of a system is an matrix corresponding to the total energy of that system.
[2] An Introduction to Hartree-Fock Molecular Orbital Theory, C. David Sherrill
[3] J. Noga and J. Šimunek, "Solving the Independent-Particle Model via Nonunitary Transformation Based on Variational Coupled Cluster Singles", *J. Chem. Theory Comput.* 2010, 6, 9, 2706–2713
[4] Sparse BLAS documentation
[5] Fortran 90/95 library for sparse matrices?

PRACE SoHPCProject Title
Efficient Fock matrix construction in localized Hartree-Fock method

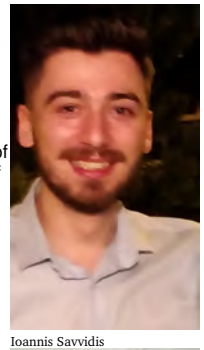
PRACE SoHPCSite
CC SAS-Computing Centre, Centre of Operations of the Slovak Academy of Sciences, Slovakia

PRACE SoHPCAuthor
Zsurka Eduárd, Eötvös Loránd University, Hungary
Ioannis Savvidis, University of Ioannina, Greece

PRACE SoHPCMentor
Ján Šimunek, Comenius University Bratislava, Slovakia

PRACE SoHPCContact
Zsurka Eduárd, ELTE
E-mail: eduard.zsurka@ttk.elte.hu
Ioannis Savvidis, University of Ioannina
E-mail: iosavvidis21@gmail.com

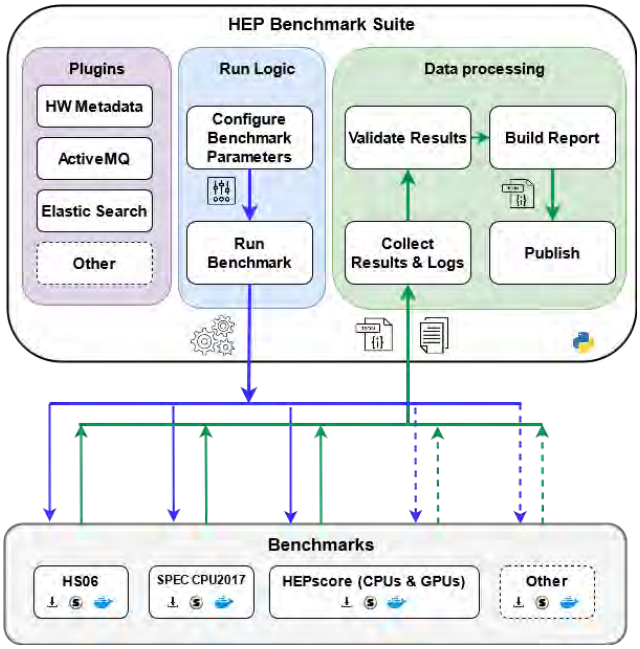
PRACE SoHPCProject ID
2108


Ioannis Savvidis


Zsurka Eduárd

HEP Benchmark Suite on HPC

María Menéndez Herrero &
Miguel de Oliveira Guerreiro



In order to evaluate new computing platforms for executing High Energy Physics Workloads, a container-based benchmarking suite was developed at CERN. The goal of the project is to understand how this workloads behave in different HPC facilities and heterogeneous systems and add new types of workloads to the suite as well as combine different benchmarking tools.

The demand for computational resources to perform calculations related to High Energy Physics (HEP) has been growing over the last few years. HEP workloads requires a large number of resources due to the enormous ammount of data and also HEP Experiments have begun adopting heterogeneous resources instead of homogenous CPU-only workloads. How can we know if the resources we have available are up to the calculation we want to perform? And how do we know how much resources we need to process a set of data? One way to find out is through Benchmarking. A HEP Benchmark Suite based on container technology has recently been developed and has only been tested on a very small number of HPC systems. Therefore, one of the aims of this project is to test this Benchmarking Suite on the Cartesius cluster (SURFsara), to test the HEP workloads on hardware types that have not been tested so far, as well as to combine efforts to explore a unified benchmarking approach that can be utilized by many areas of research.

In order to make the most of the different existing heterogeneous architectures, a collaboration between four pioneering entities in their fields: CERN, the European Organization for Nuclear Research; SKAO, the organisation leading the development of the Square Kilometre Array radio-telescope; GÉANT, the pan-European network and services provider for research and education; and PRACE, the Partnership for Advanced Computing in Europe.

The purpose of developing this Benchmark Suite is to explore the different performances that supercomputers within the reach of the four participating entities can have for the development of high-energy physics and astronomy calculations. For this reason, the main goal of this project is to develop a unified benchmarking approach for the collaboration - which will benefit all members. This task was expored by trying to combine the Unified European Applications Benchmark Suite (UEABS) with the HEP Bench-

```
#SBATCH --exclusive --hint=multithread
#SBATCH --job-name=HEP-Benchmark-suite
#SBATCH --output=HEP-result-%A-%j.out
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=user@email.com
#SBATCH --array=1-5
#SBATCH --time=72:00:00

## allocate & run the below script once on each of 5 allocated nodes.
## nodes are requested in exclusive mode use with multithreading enabled.


export RUNDIR=/tmp/HEP
export SUITE_BRANCH="master"

# example tag 'site':'my HPC...'
export BMKSUITE_TAG_SITE="my HPC site string to be included in JSON"

echo "Running HEP Benchmark Suite on $SLURM_CPUS_ON_NODE Cores"
# install suite as a module
#mkdir -p "$RUNDIR"
#python3 -m pip install --upgrade git+https://gitlab.cern.ch/hep-benchmarks/hep-benchmark-suite.git@${SUITE_BRANCH}

# run
srun -p normal --pty bash
"$HOME"/.local/bin/bmkrun --config=$HOME/my-custom-config.yml --tags --rundir "$RUNDIR"

# Copy local results to $HOME if not reporting via AMQ cp /tmp/HEP/result_profile.json $HOME/${SLURM_JOB_ID}_result.json
find ${RUNDIR} \( -name \*.json -o -name \*.log \) -exec tar -rvf "$HOME"/results-"${SLURM_JOB_ID}"-"${SLURM_ARRAY_JOB_ID}".tar {} +
```

```
     JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
10680001_1    normal HEP-Benc  mariamh  R       0:04      1 tcn1505
10680001_2    normal HEP-Benc  mariamh  R       0:04      1 tcn712
10680001_3    normal HEP-Benc  mariamh  R       0:04      1 tcn892
10680001_4    normal HEP-Benc  mariamh  R       0:04      1 tcn1037
10680001_5    normal HEP-Benc  mariamh  R       0:04      1 tcn1081
```

Figure 1: Modified script to run a Benchmark simultaneously on 5 nodes with the slurm queuing system and slurm queuing system summary when the script is running

mark Suite. UEABS is a set of currently 13 application codes taken from the pre-existing PRACE and DEISA application benchmark suites, and extended with the PRACE Accelerator Benchmark Suite. Among the 13 codes included we can find: ALYA, Code_Saturne, CP2K, GADGET, GPAW, GROMACS, NAMS, NEMO, PFARM, QCD, Quantum Espresso, SHOC, SPECFEM3D and TensorFlow.

The list of Benchmarks available in the current version of the Benchmark Suite developed by CERN, are:

- **HS06** is a subset of SPEC CPU 2006. It's currently the "standard" benchmark for CERN benchmarking, accounting, procurement, pledges, etc.
- **SPEC2017** is the newer version of SPEC CPU (2017), however it was never adopted by CERN as the official benchmark, but is available to run for comparison reasons.
- **HEPscore** is an application that orchestrates user configurable containerized HEP benchmarks.
- **DB12** and **ATLAS Kit Validation** are fast benchmarks that should not be used for performance measurements, but they are useful for quickly performance of the HEP Benchmark Suite.

The architecture of the Benchmark Suite used can be seen in the figure in the cover.

The Benchmark Suite repository can be consulted in this link gitlab.cern.ch/hep-benchmarks/hep-benchmark-suite/

The following sections describe the objectives set at the beginning of the project, as well as the methodology followed and the description of the containerization systems used by the Benchmark Suite. This will be followed by a brief analysis of the results achieved, ending with the conclusions and the projection given to the project.

Objectives

For the reasons described above, one of the mains goals of this project is to study the performance at SURFsara as well as investigate adding new types of workflows, testing HEP workloads on new hardware and trying to combine this Benchmarking Suite with PRACE benchmarking tools (UEABS).

Methodology

The methodology to be followed consists of testing different workloads on Cartesius (SURFsara) using the HEP Benchmark Suite. Firstly, in order to familiarise ourselves with the Benchmark Suite, the procedure followed consisted of running short tests of the DB12 benchmark in order to detect any possible errors that might appear. For this, it was necessary to perform a previous analysis of Singularity, which is designed for HPC with a focus on security - especially attractive as running docker would require root privileges (which are generally not available at HPC sites). Singularity is compatible with docker images, such as those found from Docker Hub. Due to several problems fixed latter to found the Singularity module at Cartesius, familiarization with singularity was also replicated at MareNostrum4 (MN4).

Then, to run the Benchmark Suite in the command line two examples can be seen in the following lines to run DB12.

```
bmkrun -c default -b db12
```

Where here, *-c default* refers to the default configuration the Benchmark Suite can be configured by the user in order to customise their calculation according to their interests. And also, an example to run the HS06 and SPEC2017 Benchmarks is included.

```
bmkrun -c <alternate
config> -b hs06
spec2017
```

The second step consisted of modifying the script that can be seen at the top of Figure 1. This script represents an example in which the Benchmark Suite is executed using the slurm queuing system to run the selected benchmark

on 5 nodes simultaneously. As can be read in the comments of this script it is written in such a way that the nodes are requested in exclusive mode use with multithreading enabled. On the other hand, in the second image in Figure 1, you can see the summary of the slurm queue system while this script is running.

Finally, in order to combine the HEP Benchmark Suite and PRACE UEABS benchmarks, we selected GPAW from the PRACE benchmarks list because it is mostly written in Python and because it was familiar to us. GPAW is a density-functional theory (DFT) Python code based on the projector-augmented wave (PAW) method and the atomic simulation environment (ASE). GPAW is freely available under the GPL license. The GPAW PRACE repository includes three examples that can be used by the user to analyse the performance of the examples for the proposed cases: carbon nanotube, copper filament and silicon cluster. To run it in in the command line:

```
srun gpaw-python input.py
```

The container that is going to generate the image necessary to run GPAW and implement it latter in the Benchmark Suite can be found in Docker Hub, where in this link you can find the instructions to generate the image with Docker.

The next step to be followed consist in running the GPAW Benchmarks in the command line when implemented in the Benchmark Suite and, finally, try to automatize it. However, in this step we have encountered some difficulties, as it required a deeper analysis of the Benchmark Suite implementation code, which we have started to do in the last few weeks, so progress in this step has not been great.

Results and Discussion

Following the methodology described in the previous section, in terms of testing the calculation performed in MN4, we initially found several dependency errors when testing at MN4 due to bugs in the HEP Benchmark Suite. On the other hand, in the SURFsara partition, Cartesius. We reported the bug details to the HEP Benchmarking team, as this was the first time it was encountered at a HPC site.

The results obtained for the benchmarks implemented in the HEP Benchmark Suite will be collected in a .json file containing all the information related to the performance of the supercomputer once the job is finished. In this project this document has not been analyzed in depth, as we have been focused on the exploration of the Benchmark Suite when executed in SURFsara and it's extension with UEABS.

As for the script shown in Figure 1, tests have been carried out for 2, 3, 4... and up to 10 nodes simultaneously. The performance for each case has not been analysed in detail, it has simply been verified that no failure has been found for running the Benchmark Suite on different nodes simultaneously. In the second image in the Figure you can see how this calculation is being performed on the partition named *normal* and how by the node label each node is different. In this step no significant bugs have appeared other than problems in saving the files that could be easily solved by modifying the directories in the script.

As mentioned above, one of the main objectives of this project was the implementation of the Unified European Application Benchmark Suite in the Benchmark Suite developed by CERN in order to automate it. As a representative example GPAW has been selected and the benchmarks included in UEABS have been run within the locally generated container image successfully after analyzing the output of GPAW container. However, due to the time available, the complexity of the code and the HPC containerization process, this task has not been completed. On the other hand, the containers needed to run the applications were found and tested, also, it have been tested that the image of GPAW-OpenMP is compatible with Singularity so the last step needed is just the implementation in the CERN Benchmark Suite, but this is not a trivial task as we could experience.

Conclusions and Outlook

Developing and completing this Benchmark Suite can be very useful to determine the performance of supercomputers in a repeatable (thanks to containerization) and easy way. This is because it is mainly focused on the automation of different Benchmarks, so once the benchmark is implemented in this HEP Benchmark Suite.

To conclude, among the task fulfilled during these two months project we can find the exploration of the Benchmark Suite in SURFsara, which includes the addition of workflows in different nodes simultaneously as well as the test of HEP workloads in this new type of hardware. Although we did not fulfill all of the goals the task of combining the PRACE benchmarking tools with this Benchmarking Suite, however, given the progress we have made in this regard, we are very hopeful that this will soon be achieved. On the other hand, GPAW is a MPI workload in HPC. HEP Benchmark Suite is not yet adapted for taking advantages of these features of HPC, thus requires significant development.

PRACE SoHPCProject Title

Benchmarking HEP workloads on HPC facilities

PRACE SoHPCSite

CERN, Switzerland
SURFsara, Netherland

PRACE SoHPCAuthors

María Menéndez Herrero &
Miguel de Oliveira Guerreiro

PRACE SoHPCMentor

David Southwick, CERN, Switzerland

PRACE SoHPCContact

Miguel de Oliveira Guerreiro
E-mail:
oliveira.guerreiro@tecnico.ulisboa.pt
María Menéndez Herrero
E-mail:
mariaherrero9717@hotmail.com

PRACE SoHPCSoftware applied

Cartesius, Surfsara
HEP Benchmark Suite
Singularity
Python
GPAW

PRACE SoHPCMore Information

www.userinfo.surfsara.nl
www.gitlab.cern.ch/hep-benchamarks/hep-benchmark-suite
www.repository.prace-ri.eu/git/UEABS/ueabs/

PRACE SoHPCAcknowledgement

We would like to thank our mentor David Southwick for his guidance throughout the duration of the project. We would also like to thank the PRACE Summer of HPC for the opportunity to carry out this project.

PRACE SoHPCProject ID

2109

María Menéndez Herrero

Miguel de Oliveira Guerreiro

HEP Data Processing at HPC

Carlos Cocha & Andraž Filipčič

The High Energy Physics (HEP) community typically employed High Throughput Computing (HTC) type of facilities for data processing and physics analyses of data coming from the Large Hadron Collider (LHC). For that reason, the goal of this project is to use several types of workloads to take mock-ups, scale them out and evaluate their effectiveness at an HPC scale.

Introduction

HTC type of facilities are traditionally employed by the HEP community for the purpose of the LHC data processing and various types of physics analyses. Furthermore, with recent convergence of AI and HPC, a single but modular and flexible type of facility could replace single-purpose based environments in the near future.

Furthermore, LHC type of workloads are mostly data-driven, which means that a lot of data has to be ingested and substantial amount of output produced. However, the handling of input and output using thousands of nodes becomes a bottleneck.

Nowadays, part of the HEP community evaluates various proof-of-concept designs and how this data flow works. The idea for this project is to scale different workloads, but still both data driven, and understand the limitations of the existing model and observe peculiarities of HPC systems under heavy dataflow load as well. In this sense is necessary to benchmark these facilities in order to evaluate and rank system performance and find its limitations.

For that reason, the goal of this project is to take various I/O driven applications/mock-ups, similar to the I/O driven physics analyses that could employ some type of Machine Learning or Deep Learning, scale them out at an HPC facility and evaluate their effectiveness under heavy dataflow load.

Methodology

The HPC system used for the development of the project is the **CSCS Grand Tavé**[1] located in the Swiss National Supercomputing Centre (CSCS) in Switzerland. It has a theoretical peak performance of 436.63 TFlops and maximum number of nodes equal to 164. The Scratch filesystem which is the shared storage connected via infiniband interconnect to the system is used to measure the cluster performance.

The first part of the project was focused in get familiarized with the Grand Tavé system. Later, mock up tests using FIO[2] and IOR were generated to obtain and visualize performance metrics (e.g. bandwidth).

FIO

Flexible I/O tester (FIO)[2] is an open source synthetic benchmark tool that can generate various I/O type workloads. The biggest difference comes from how the data is being read from the disk, see Figure 1.

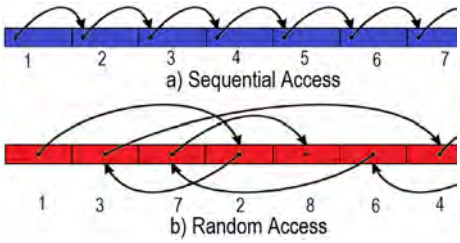


Figure 1: Types of I/O access patterns.

In sequential access, the system reads the information to the file sequentially, starting from the beginning of the file and proceeding step by step. On the other hand, during random access the system can read information anywhere in the data file.

Inside a FIO job file is possible to set-up the different parameters of the workload one wants to simulate, such as: the block size (bs), the number of clones of the job (njobs), the filesystem to be tested (which in this case is the Scratch filesystem of Grand Tavé) and the type of the I/O pattern.

Then, submitting the job file through Slurm is possible to plot the metrics as a function of the number of nodes and other parameters (e.g. njobs). If we scaled out to a high number of nodes and high number of njobs the peak of performance for a certain workload configuration can be visualized.

Sequential read

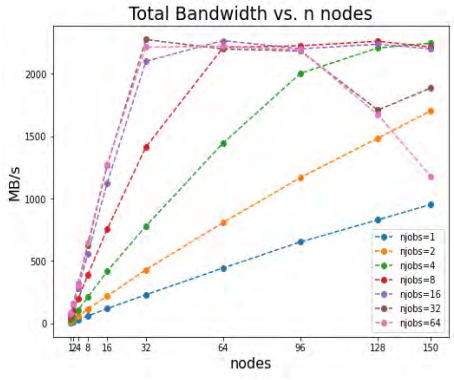


Figure 2: Bandwidth for sequential read with bs = 4K configuration.

For example, Figure 2 shows the total bandwidth utilization for different number of nodes and number of jobs in a sequential read configuration with a block size of 4 kilobytes (4K) where a maximum bandwidth of 2.2 GB/s is obtained.

Random read

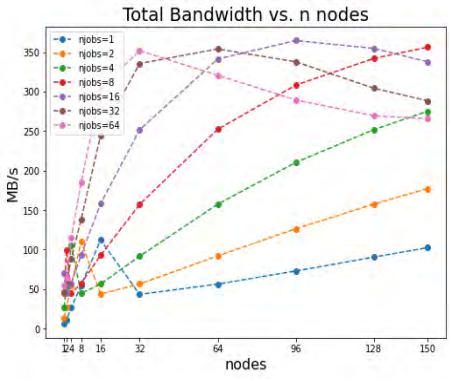


Figure 3: Random read with bs = 4K.

Testing a random read workload using the same sequential read configuration with a block size of 4K, as we expected, the random read is slower than the sequential read with a bandwidth peak of 0.36 GB/s, see Figure 3.

Now, the metric under different block sizes are tested and the results, visualized in 3D, are shown in Figure 4 and Figure 5 for a sequential read and random read configuration respectively. Two important facts are observed. First, the bandwidth increases at the same rate as the block size does. Second, roughly the same amount of I/O operations per second.

Then, the maximum bandwidth results for different block size configurations are summarized in Table 1. In average, the sequential read is 6 times faster than the random read configuration. But, random access has the advantage that the system knows where the data is stored and can find it more easily (using indexing).

Bandwidth (GB/s)			
Mode	4K	8K	16K
Seqread	2.27	4.34	8.30
Randread	0.36	0.71	1.37

Table 1: Maximum bandwidth as function of the block size.

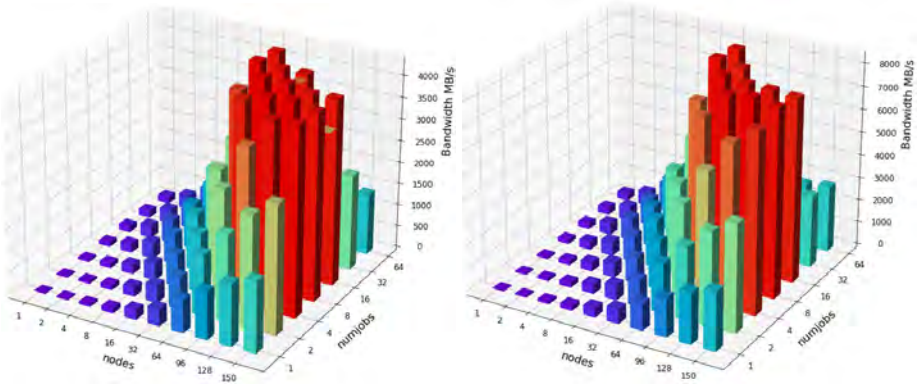


Figure 4: Sequential reads 3D plots. Left: bs = 8K. Right: bs = 16K

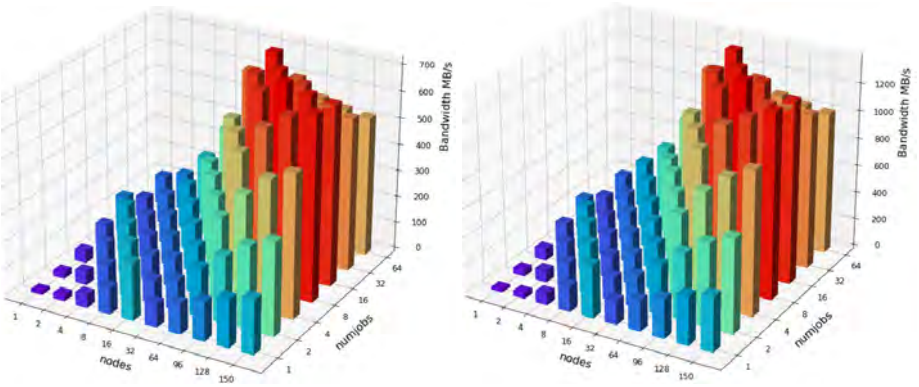


Figure 5: Random reads 3D plots. Left: bs = 8K. Right: bs = 16K.

IOR

IOR is a parallel IO benchmark that can be used to test the performance of parallel storage systems using various interfaces and access patterns.[3]

The main difference between FIO and IOR is that while FIO runs multiple processes using fork() function, IOR uses MPI to handle multiprocessing. This causes no difference in the test results, as it is simply a different way to start the processes. At the same time, it also enables communication between processes, which means that it compiles all the test results into a single log file rather than having a separate log file for each process, making the interpretation of results much easier.

Another notable difference between the two tools is that FIO is able to quickly generate random files that it then reads, while IOR has to go through the process of actually fully writing all the files it requires before then reading them.

With IOR we generally used larger block sizes than with FIO, it seemed to run a lot better that way, and was also faster, which was fairly useful as we were running short on time.

Sequential read

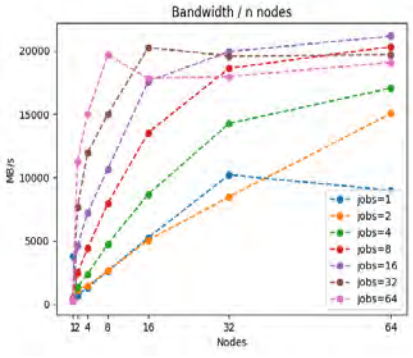


Figure 6: Bandwidth for IOR sequential read, block size = 16MB

As seen in Figure 6, the curve is roughly similar to the ones in FIO (Figure 2). It spikes faster and reaches a higher bandwidth, this is both likely due to the increased block size.

In Figure 7 we see that IOPS are much lower compared to FIO, due to the larger block size meaning less transfers need to be done.

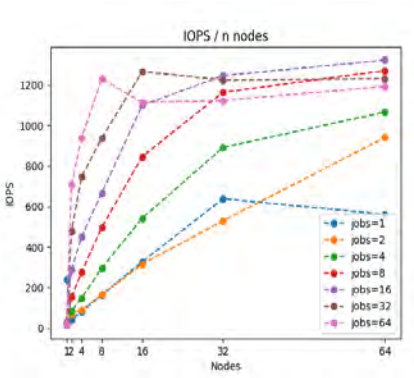


Figure 7: IOPS for IOR sequential read, block size = 16MB

Random read

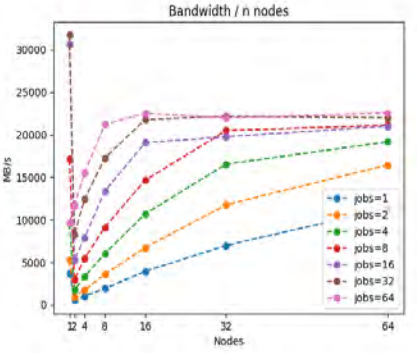


Figure 8: Bandwidth for IOR random read, block size = 16MB

In Figure 8 we see a high spike right at the start, when only a single node was used. We suspect this may be due to the random file access for processes not bypassing the memory caching, but we are uncertain why this did not appear with sequential reads (Figure 6). The random reads should be much slower than sequential reads, we're not certain why they performed so well here, which bears further investigation.

Caching

Memory caching is a big problem when benchmarking filesystems. If the files that a node is reading are small enough to all fit within its memory, then after the first time it reads the files, it will simply access the data from its own memory rather than accessing the disk.

FIO invalidates the cache by default, which seemed to work well. But IOR requires some toggles to be turned on to use workarounds.

There are several ways to avoid this problem:

- Ensure files are larger than the node's memory capacity, meaning it will be unable to fit them all in its memory and will have to repeatedly access the disk.

  Due to time constraints, we were unable to properly try this option, as our tests already had a long runtime, and sufficiently increasing the file sizes would make the tests take too long.

- Randomize which nodes read which files. This should make it so in repeated tests, the files read by each node are shuffled, meaning it will have to access different files each time. The node should eventually run out of memory capacity so when it would read the same file later on, it will not be cached anymore.

  We're uncertain how well this worked with our tests since multiple processes ran on the same node, meaning if they read each other's files, those could potentially still be cached in that node's memory from earlier.

References

[1] CSCS (2021). GRAND TAVÉ. Retrieved from: https://www.cscs.ch/computers/grand-tave/

[2] Axboe, J. (2021). fio I/O tester. Retrieved from: https://fio.readthedocs.io/en/latest/fio_doc.html

[3] IOR (2021). ior. Retrieved from: https://ior.readthedocs.io/en/latest/

PRACE SoHPCProject Title

High Throughput HEP Data Processing at HPC

PRACE SoHPCSite

CERN, Switzerland

PRACE SoHPCAuthors

Carlos Cocha,
University of Padova, Italy
Andraž Filipčič,
University of Ljubljana, Slovenia

PRACE SoHPCMentor

Viktor Khristenko, Maria Girone;
CERN, Switzerland

PRACE SoHPCCo-mentor

Sadaf Roohi Alam;
CSCS, Switzerland.

Carlos Cocha

Andraž Filipčič

PRACE SoHPCSoftware applied

fio, IOR, Slurm, Docker, Jupyter

PRACE SoHPCAcknowledgement

We would like to extend our utmost gratitude to our mentor/co-mentors, as well as all the very helpful staff at CSCS. Also thank the coordinators of the Summer of HPC 2021, and PRACE, for providing us with this unique opportunity, but also an exceptional learning-curve.

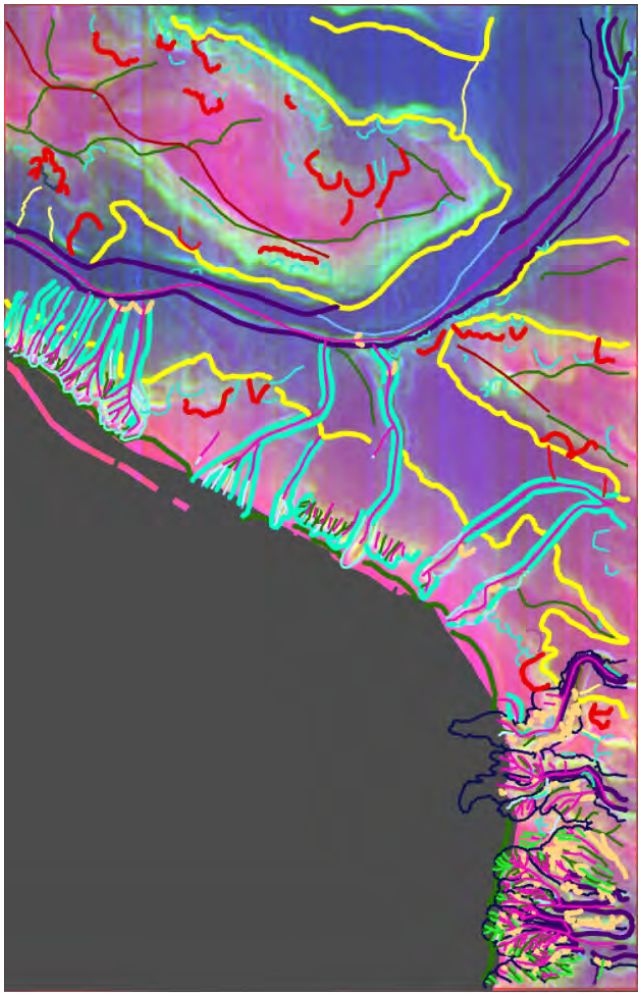
PRACE SoHPCProject ID

2110

Recognition and classification of subsea structures using artificial intelligence and high-performance computing

Machine Learning How to Sea

Mario Gaimann & Raska Soemantoro



Understanding subsea reliefs is typically a task given to specialist geologists manually labelling each area. With artificial intelligence and high performance computing, we propose a method to automate this process entirely.

Everyday, when you get up, take a hot shower and text "Good morning", you probably do not realise how important the seabed safety is for these activities. This relation may seem odd, but in our daily lives we rely on a range of hidden infrastructure deep on the seafloor, without noticing. In fact, the gas used to boil your water and the bits and bytes of your text may have whizzed through submarine pipelines and cables. Even if this infrastructure is deep under the sea, there are potential hazards threatening their safety[1] (and hence, the safety of your hot shower!).

The most important hazards are so-called geohazards. They include formations such as submarine trenches, canyons[2] or seamounts, which can cause earthquakes and submarine landslides.[3] Because of their impact on infrastructure, marine life, and coastal safety, marine geologists cruised the coastal sea in research vessels and recorded the depths of the seafloor using sonar devices. This was done in the Italian MaGIC project[4] (Marine Geohazards along the Italian Coasts).

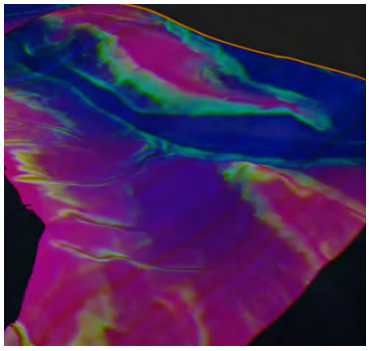


Figure 1: 3D Visualization of the submarine landscape off the coast of Calabria, Italy.

Based on the depths at various positions, the geologists obtained the shape of the underwater landscape, and recorded it in so-called bathymetric maps (bathymetry is the study of floors of seas, lakes and other water bodies). They were then able to recognise relevant seabed structures and geohazards, and sketched them in the map. Together, this lead to a composite map of depth and geological features drawn as lines, telling us where geohazards are located. There are some catches however: this submarine relief recognition by geologists is a tedious, time consuming process. Geologists might also disagree about the exact location of certain geohazards. And of course, as to err is human, this manual procedure is prone to errors, and certain geohazards could be potentially left out. Because of these reasons, we need an automated technique, which promises to be faster, objective and more reliable. And exactly this was the challenge that we tackled in our Summer of HPC project: to develop an automated technique that is able to recognise geohazards on a previously unseen bathymetric map. What techniques can we use to automatically recognise these geohazards?

Building an AI geologist

To solve this problem, we have to ask ourselves how human geologists recognise geohazards. With years of training and knowledge, they are able to tell which areas of a submarine landscape are geohazards. They recognise geohazards by looking at certain characteristics of the bathymetric map: the depth

of the sea, how the depth changes (the slope of the subsea terrain), and other more sophisticated parameters (topographic indices). For example, a submarine canyon is characterised by a deep, channel-like region, surrounded by higher regions, and there is a steep transition between these regions.

Analogous to how an aspiring geologist learns what makes a geohazard, we would ideally like to train a machine to perform this recognition. The solution lies in a particular field of computer science, which has dealt with such "intelligent" machines for many years: Artificial Intelligence (AI)!

For our geohazard learning problem, we use deep neural networks (DNNs), which can be imagined as an artificial brain. A DNN consists of multiple layers of artificial neurons, which are inspired from real, biological neurons: they are linked to other neurons and fire once they receive certain signals. By adjusting the connections between the artificial neurons, the DNN is able to learn. Overall, the sum of all neural connections can be described by a complex mathematical model with millions of parameters, which are optimised step-by-step. This approach typically requires a large amount of data (which we have thanks to the Italian MaGIC project) and a large amount of computational power to process the data (which we have thanks to the Summer of HPC!). With the MARCONI100 supercomputer,[5] operated by our host institution CINECA in Italy, we combine the power of high performance computing (HPC) and AI. For this purpose, we used graphical processing units (GPUs) and a corresponding programming technique called CUDA, which significantly accelerated the training of our "AI geologist." In the following we show you which model we chose and how our data flows through our machine learning pipeline.

Case study: Detecting geohazards in the Italian coastal sea

While there have been initial studies for the automated recognition of seabed structures,[6,7] our large-scale deep learning approach can be considered novel in the field of submarine geology. This project started as a blank canvas, meaning that we had to come up with the right software tools and strategies for a broad range of tasks, such as preprocessing raw data, designing and training a machine learning model, as well as performing evaluations. We were excited to take on this challenge and started from zero by processing a set of raw X, Y and Z coordinates forming a bathymetric map of the Ionian continental margin, off the coast of Calabria in southern Italy. In the following, we describe the 5 steps that our method comprises in detail.

Two R-CNNs for recognition and classification of subsea structures

In the first step of our method, the bathymetric map is visualized using GIS (Geographic Information Systems) software, where secondary features such as slopes, curvatures and topographical indices are derived from depth data. This outputs three separate maps, each describing these selected secondary features, as shown in fig. 2.

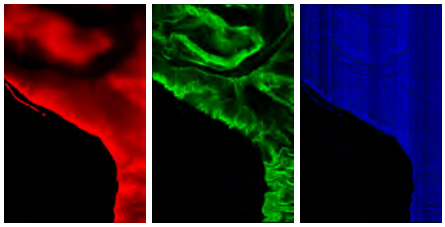


Figure 2: Three geological feature maps plotted in red, blue and green. Merging them into one image yields a combined RGB map, shown in the map on the first page (without lineaments).

The maps are then combined into one composite map using the conventional colour channel system RGB; Red, Green, and Blue. Essentially, we treat the measurement data like an image. This is an ideal strategy as like images, bathymetric measurements have some inherent hierarchy in where and how features are placed.

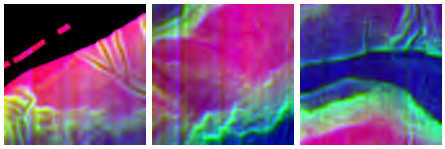


Figure 3: Small windows cut from the full composite map shown on the first page (without lineaments).

Secondly, the composite map is cut into smaller windows as shown in fig. 3. For each window, we employ an algorithm called *Selective Search*[8] to localise each feature. This algorithm uses a variety of complementary image partitioning methods to deal with as many image conditions as possible. The output of this stage is a list of possible areas, or *bounding boxes* where features might exist, as seen in fig. 4. As the manually labelled data is available through the larger map (shown on the first page), it is possible to match a label to each bounding box.

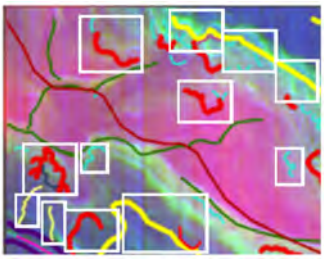


Figure 4: Selective search results from a single window as shown in fig. 3. Each subimage framed by a white boundary box contains a geologically relevant region. Colored lines here (and in the first page composite map) describe manually sketched geological features such as submarine canyons, which are used as labels for our dataset.

The third step is where we finally train the neural network model; we use each bounding box as the training images and the labels provided as the desired output. This is a well-known task in the field of computer vision, which is typically solved using the supervised learning of so-called Convolutional Neural Networks (CNNs).[9] These are similar to traditional neural networks, but a CNN convolves the input in the first layer similar to how neurons in the visual cortex react when given visual stimulus.[10] As we have already done the feature regions in the previous step, this model is therefore called a Region-based Convolutional Neural Network (R-CNN).[11] And as the type of subsea reliefs are plenty and can be quite similar to each other, we employ two models, trained in parallel. The first model is the feature existence model that is trained to recognise whether an area contains a feature or not, and the second is the feature type model that further classifies each area into a single type of relief. Using this architecture, the model is trained to correctly assign the correct label (i.e., the type of subsea relief) to a part of the composite map. To achieve high accuracies, we adjust the parameters (so-called hyperparameters) of our models. At this stage, we have completed the training section of the

```
Epoch 1743/5000
50/50 [==============================] - ETA: 0s - loss: 0.1939 - accuracy: 0.9419
Epoch 01743: val_loss did not improve from 0.03967
50/50 [==============================] - 19s 382ms/step - loss: 0.1939 - accuracy: 0.9419 - val_loss: 0.3019 - val_accuracy: 0.8750
Epoch 1744/5000
50/50 [==============================] - ETA: 0s - loss: 0.1696 - accuracy: 0.9394
Epoch 01744: val_loss improved from 0.03967 to 0.03893, saving model to ./models/BIN_RCNN_HYP_1/BIN_RCNN_HYP_1-C11.hdf5
```

Figure 5: Snapshot of the training process of our feature existence model using the Tensorflow framework.

framework, which concerns converting labelled digital maps into training data and the training itself. This is shown in fig. 6.

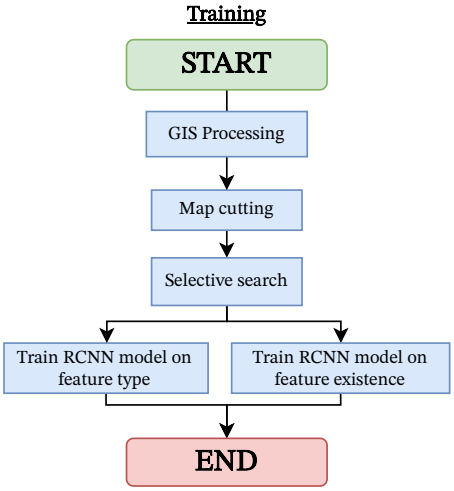


Figure 6: Flowchart of model training.

The final step is the evaluation stage, in which we combine both feature type and existence models. This stage is visualised in fig. 7. Once we cut the map and perform selective search, we pass images with regions of interest to the feature existence model. If no feature existence is predicted, the image is discarded. If the first model predicts that a feature might exist in the given image, it passes the image to the feature type model. The second model predicts the expected class of the given image, and then the predicted class is then stored together with the image, giving us a collection of geological images and their predicted labels.

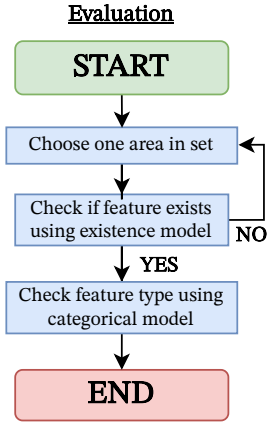


Figure 7: Flowchart of model evaluation.

Predicting geohazards with high accuracy

Our work resulted in a fully automated seabed relief classification framework, using two R-CNN models. The training section of the pipeline contains steps to convert the map into learnable data for the two models, and the evaluation section contains steps on how to integrate both models into a fully automated system. Both the feature existence model and the feature type model achieved accuracies of over 90% as shown in fig. 5, which allowed us to make predictions with high confidence.

Conclusion

In this project, we set out to independently develop an AI-based method to automate the classification of subsea reliefs. Our method comprises 5 steps; pre-processing of geological data, performing selective search, training two R-CNN models, and evaluating the predictions. Doing research in a domain that intersects between machine learning and marine geology is an exciting challenge to take on for a small team of a physicist and an engineer, but we are glad to have gone through such a rewarding journey.

References

[1] Francesco L. Chiocci, Antonio Cattaneo, and Roger Urgeles. *Seafloor mapping for geohazard assessment: State of the art*. Mar. 2011.

[2] Silvia Ceramicola et al. "Submarine canyon systems along the Ionian Calabrian margin, Central Mediterranean Sea". In: *Submarine canyon dynamics in the Mediterranean and tributary seas - An integrated geological, oceanographic and biological perspective. CIESM Monograph 47* April (2015), pp. 157–163.

[3] Kyoji Sassa, Paolo Canuti, and Yueping Yin. *Landslide science for a safer geoenvironment*. Vol. 1. Springer International Publishing, Jan. 2014, pp. 1–486.

[4] Daniele Casalbore et al. *The Italian MaGIC Project. Hydro International. https://www.hydro-international.com/content/article/the-italian-magic-project.* 2011.

[5] CINECA. *MARCONI100. https://www.hpc.cineca.it/hardware/marconi100*. 2021.

[6] Khaira Ismail, Veerle A.I. Huvenne, and Douglas G. Masson. "Objective automated classification technique for marine landscape mapping in submarine canyons". In: *Marine Geology* 362 (Apr. 2015), pp. 17–32.

[7] Michelle Linklater et al. "Techniques for classifying seabed morphology and composition on a subtropical-temperate continental shelf". In: *Geosciences (Switzerland)* 9.3 (Mar. 2019), p. 141.

[8] J. R.R. Uijlings et al. "Selective search for object recognition". In: *International Journal of Computer Vision* 104.2 (2013), pp. 154–171.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[10] Grace W. Lindsay. "Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future". In: *Journal of Cognitive Neuroscience* (Feb. 2020), pp. 1–15.

[11] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Nov. 2014, pp. 580–587.

PRACE SoHPC Project Title

Automated classification for mapping submarine structures by artificial intelligence strategies

PRACE SoHPC Site

CINECA, Italy

PRACE SoHPC Authors

Mario Gaimann, Ludwig-Maximilians-Universität München, Germany
Raska Soemantoro, The University of Manchester, United Kingdom


Mario Gaimann


Raska Soemantoro

PRACE SoHPC Mentors

Silvia Ceramicola, OGS, Italy
Veerle Huvenne, NOC, UK
Massimiliano Guarrasi, CINECA, Italy

PRACE SoHPC Contact

Mario Gaimann, Ludwig-Maximilians-Universität München
E-mail: mug502@alumni.york.ac.uk
Raska Soemantoro, The University of Manchester
E-mail: raska_soemantoro@outlook.com

PRACE SoHPC Software applied

Python, Tensorflow, OpenCV, CUDA, Numpy, Pandas, Numba, SAGA GIS, ArcGIS, Meshlab, Blender

PRACE SoHPC Acknowledgements

This research would have not been possible without the support of PRACE and the organisers of Summer of HPC, notably Dr. Leon Kos. We thank OGS (Istituto Nazionale di Oceanografia e di Geofisica Sperimentale) for providing the maps, obtained through the framework of the MaGIC (Marine Geohazards along the Italian Coasts) project, funded by DPC (Dipartimento della Protezione Civile). We also thank Francesca Delli Ponti of CINECA for aiding us with the Blender visualisation of the subsea map.

PRACE SoHPC Project ID

2111

Visualisation and Anomaly Detection of A Supercomputer

David Mulero Pérez, Sepideh Shamsizadeh

This project aims to look for a way to create a virtual replica with which we can visualise all the information we have from the Marconi 100 supercomputer and use that data to detect anomalies. And proposing a deep learning(DL) model for anomaly detection.

High-performance computing (HPC) makes it easier to perform large amounts of computation in projects of all kinds. For this purpose, supercomputers consisting of hundreds of nodes are used, each consisting of many cores. The need for supercomputing is growing, and new centers are being built, often larger and more powerful than the previous ones. Due to the number of components, monitoring and maintaining the state of supercomputers is not an easy task.

Therefore, solutions to this problem are being sought. Finding a way to control the states and to predict possible errors. In addition to this, supercomputers do not have the same architecture, although they are often similar, and a specific solution must be designed to work with this computing center.

Anomaly detection is the process of accurately analyzing time-series data in real-time, identifying unexpected items or events in data sets, which differ from the norm (no anomalies). Also, it can be an extreme case of an unbalanced supervised learning problem, as the vast majority of data generated by real supercomputers is, by definition, normal.

In today's world, the issues concerning their maintenance are increasing by rapidly becoming larger and complex the High-Performance Computing (HPC) systems. Anomaly detection can help pinpoint where an error occurs, enhancing root cause analysis and quickly getting tech support before reaching a critical state.[1]

Method and Results

Method for Visualisation

First of all, let's look at the architecture of the Marconi 100, which we are going to work with. Marconi 100 is made up of 55 racks. Each rack has 20 nodes inside it. And each node is composed of 2 x 16-core processors and 4 GPUS. As a curiosity, with its 32 Pflop/s, it was the ninth largest supercomputer in the world in 2020 and uses the Red Hat Linux distribution as its operating system. To create this visualisation we used the VTK (Visualization Toolkit)[2] package for Python and Pandas to handle the data in table form. The data we need is extracted from an ExamonDB database. The database has information since the supercomputer was started. Data such

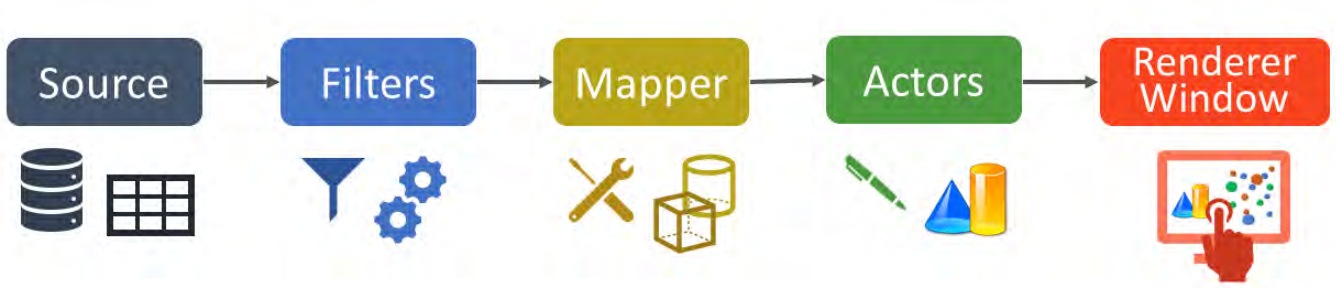


Figure 1: VTK Pipeline

as the temperatures of both the CPU cores and the GPUs. The power, voltage and memory consumed by each CPU. And one of the most important data for us is the status, which indicates whether the node is working properly or has a problem. A value of 0 means that there are no problems and any other value means that there is a problem.

Since it is a very large amount of data and we want to facilitate its management, it is stored in a different file for each node, that means 980 files. We name these files by their node name. This is always the rack number and the identifier of the node within the rack. For example, node 15 in rack 208 would have the following name r208n15.

Creating the 3D model within VTK is easy thanks to the vtkSTLReader class, which allows us to import 3D files in STL format. In this way, I have been able to use the Marconi 100 3D models that I was provided with by the CINECA visualisation team.

VTK library offers different classes that are responsible for carrying out different parts of the process. In short, the visualisation pipeline consists of the following parts: Sources, used to read raw data. Filters can then be used to modify, transform and simplify that data. Mappers are responsible for creating tangible objects with the data. And at the end of this pipeline are the Actors, which are responsible for encapsulating these objects in a common interface that has many freely modifiable properties. These Actors are the ones that are passed to our renderer to show them in the scene. A diagram is shown in figure XX. In addition to all this, other much more specialised classes can be created that allow us to develop a more interactive environment, among other things. In my case, I have used this to create keyboard shortcuts that I use to move around the time axis when displaying data.

The reading and processing of data requires a lot of computing time, so it is interesting to parallelise the process. And in this case it is very easy to do so, as the data for each node is in a separate file and inside Python we store it in an individual table. So we have used an instance of ProcessPoolExecutor from Python's concurrent.futures module, which creates as many threads as nodes we have to read and handles them automatically depending on the capacity of our CPU. Doing this, I have managed to triple the loading speed in my pc that only has 6 cores. Using a Marconi 100 node can reduce the loading time by up to 10 times.

Figure 2: The 3D model within VTK

Result for Visualization

In figure 2 we can see an example of what has been the result of the creation of the virtual replica of M100. You can see a 3D model of the racks, the walls, and even where the air conditioners are located. Each node has the following information: a text with the power it is using (this text turns red if the power is too high), a square representing the status of the node (green if everything is working properly and red otherwise). In addition, on the left side of each node, we can see a heat map with all the temperatures of the cores of the node and the top 16 dots refer to the temperatures of processor 1 and the bottom 16 dots refer to the temperatures of the second processor.

Furthermore, all this data can be displayed in 15-minute intervals using keyboard shortcuts. To do this, callbacks that VTK allows have been used. When a callback is executed we move the time forward or backward by 15 minutes and update the data of the objects that are being represented in the scene (Sources objects). As we can see, no textures are used and the objects have very few polygons, otherwise, it could consume a lot of resources if run on some devices. In this case, we have to give priority to usability and we don't care so much about how it looks aesthetically. It could be optimized for use on the Marconi 100 nodes and get much better performance. In this case, a lot more details could be added.

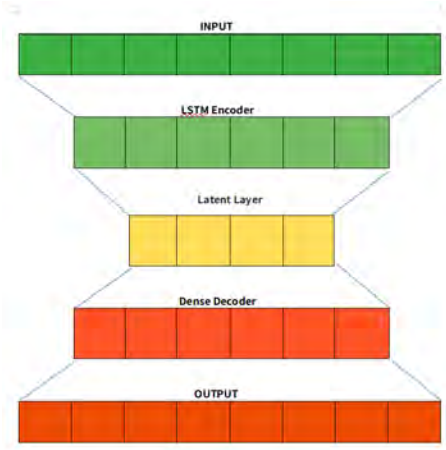


Figure 4: Structure of autoencoder

Anomaly Detection

Autoencoder is an unsupervised learning technique capable of efficiently compressing and encoding data and reconstructing the data from the reduced encoded representation to a representation close to the original input as possible. For using Autoencoder for anomaly detection, we first train an Autoencoder on normal data, then take a new data

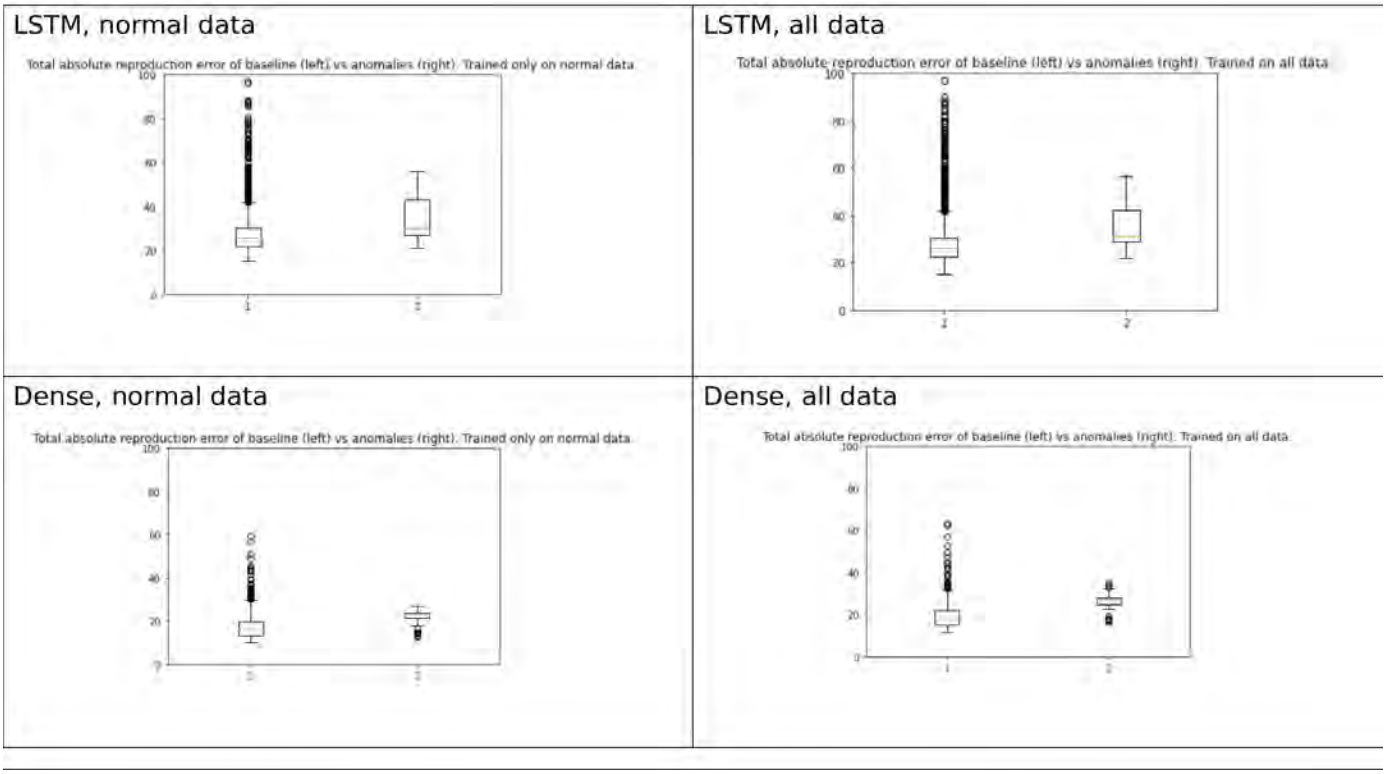


Figure 3: Compration Dense and LSTM model.

point and reconstruct it using the Autoencoder. We used the data of combination of 5 nodes.

Result of Anomaly Detection

Training and evaluating machine learning models usually require a training set and a test set. In most cases, train and test splitting are done randomly by taking 80% of the data as train data and using the rest for the test, unseen by the model. In the case of time series, we cannot choose random samples and assign them to either the test or the train because it makes no sense to use the future values to determine past values.

The first 80% is given as training data and the remaining 20% as test. Time series data must be transformed into a structure of samples with input and output components before it can be used to fit a learning model. To embed time series data to our network we used TimeSeriesGenerator.

We selected two autoencoder models for training our data. Dense model which, All layer of encoders, latent, and decoders were dense layer. And The LSTM model that the encoders and latent layer were LSTM layer and for decoders were dense layer. The activation function for each layer was SELU and optimization function was RMSprop(1e-3).

We train both the Dense and LSTM models on all data and normal data to compare their performance. As you can see the results of total absolute reproduction error of baseline and anomalies for Dense and LSTM models on all and normal data are shown in Figure 3. What we want to see the higher reproduction error on anomaly and low reproduction error on normal data. The plot with high separation between reproduction error for anomaly and normal data is the best. So, the best model is the Dense on normal data. We are working on a clean data set. We figure out if we are using the LSTM, we take count the temporal dependencies. So, as Dense model is better than LSTM we can say that there is no temporal dependencies between folds.

References

[1] Borghesi, Andrea and Molan, Martin and Milano, Michela and Bartolini, Andrea (2021). Anomaly Detection and Anticipation in High Performance Computing Systems. IEEE Transactions on Parallel and Distributed Systems.

[2] Schroeder, W. J. (1999). The VTK user's guide. Kitware.

PRACE SoHPCProject Title

Combining Big-data, AI and 3D visualization for datacentre optimization

PRACE SoHPCSite

CINECA, Bologna, Italy

PRACE SoHPCAuthors

David Mulero, University of Alicante, Spain
Sepideh Shamsizadeh, University of Padova, Italy

PRACE SoHPCMentor

Andrea Bartolini, University of Bologna, Italy
Martin Molan, University of Bologna, Italy

PRACE SoHPCContact

David, Mulero Pérez, University of Alicante
Phone: +34 722 52 99 38
E-mail: davidmuleroperez@gmail.com
Sepideh,Shamsizadeh, University of Padova
Phone: +3495736491
E-mail: sepideh.shamsizadeh@studenti.unipd.it

PRACE SoHPCSoftware applied

Python

PRACE SoHPCAcknowledgement

We are thankful to our mentors from the University of Bologna and CINECA for their help, guidance and support throughout this project

PRACE SoHPCProject ID

2112

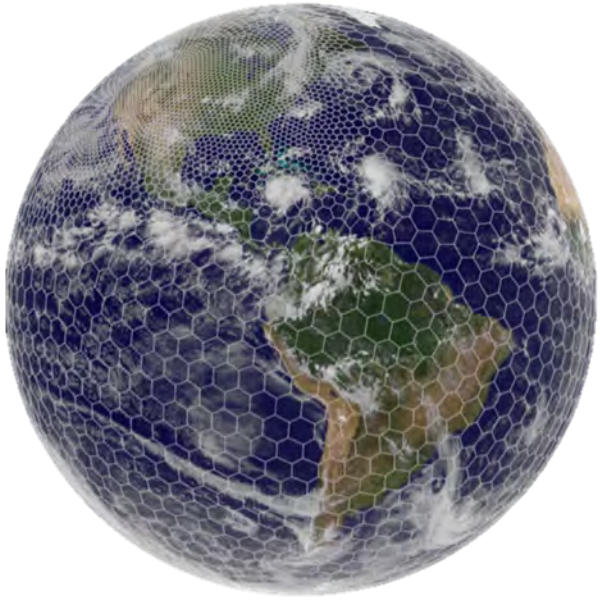

David Mulero


Sepideh Shamsizadeh

Scaling, Tiling and the global atmosphere

Eschenfelder, Jonas
Schoder, Carla Nicolin

MPAS Atmosphere was designed with scalability in mind. Its use of Voronoi tessellations allows for scaling in model resolution and it's focus on parallel computing allows for scalability in performance. We tested this for both global and regional models and found good strong and weak scaling. We also developed visualisation tools that allow for easy conversion of the raw data to beautiful animations using Python.



a

[a]Image modified from https://mpas-dev.github.io/

What shapes can tile a sphere? This seemingly abstract geometry problem becomes very applied as soon as you start thinking about how to split up the world. We are used to the latitudes and longitudes, seemingly splitting up the world into rectangles. So it seems only logical to use this system when setting up atmospheric models. But while it is a good system to pinpoint where you are on this planet, it comes with many problems for modelling. First, the distortion of these seemingly regular rectangles (something that has plagued cartographers for a long time) makes it impossible to create mesh with uniform resolution spanning the entire globe. Secondly, as regional weather not only depends on large scale patterns but also small scale regional complexities, we often need a high resolution over the region of interest but only a lower resolution everywhere else. When working in rectangles this will end up creating harsh boundaries along the edge of regions with different resolutions, leaving the modellers with artefacts that are hard to clean up. To solve these issues the developers of the Modelling Predictions Across Scales (MPAS) Atmosphere model, resorted to using a similarly abstract mathematical object, the Voronoi Tessellation. Voronoi Tesselations create meshes, where, instead of defining an area by its boundaries, it is defined by a central point and each cell spans the area that is closest to it's central point. When the points are spaced regularly, a hexagonal mesh covering the entire globe is created without any distortions. Through the Voronoi mesh, irregular cell shapes are possible and this allows for resolutions to vary smoothly without creating model artefacts. Another advantage of MPAS is that it was built with parrallel computing in mind, promising good performance when run on HPC systems. Be it numerical weather predicitions or the modelling of the impact of climate change, fast and reliable atmospheric models are needed.

In this project we tested MPAS Atmosphere on ARCHER2, the UK's new national supercomputer, to investigate its performance and scalability. We also looked at how to best visualise the outputs, as NCAR, the developers behind MPAS, recently announced stopping of development of their own plotting language NCL in lieu of moving to Python.

Scaling and Performance

We used the most recent release of MPAS atmosphere, available via gitHub here: https://github.com/MPAS-Dev/MPAS-Model. There are two use cases for MPAS, it can either be run globally to model the entire Earth's atmosphere at a time or regional cuts can be made to only model that areas atmosphere. We tested both and the results are shown below.

Global Models

To test the global model we ran the Jablonowski-Wiliamson baroclinic wave[1] test. This is a standard idealised case often used to test different atmospheric models. While not as complex

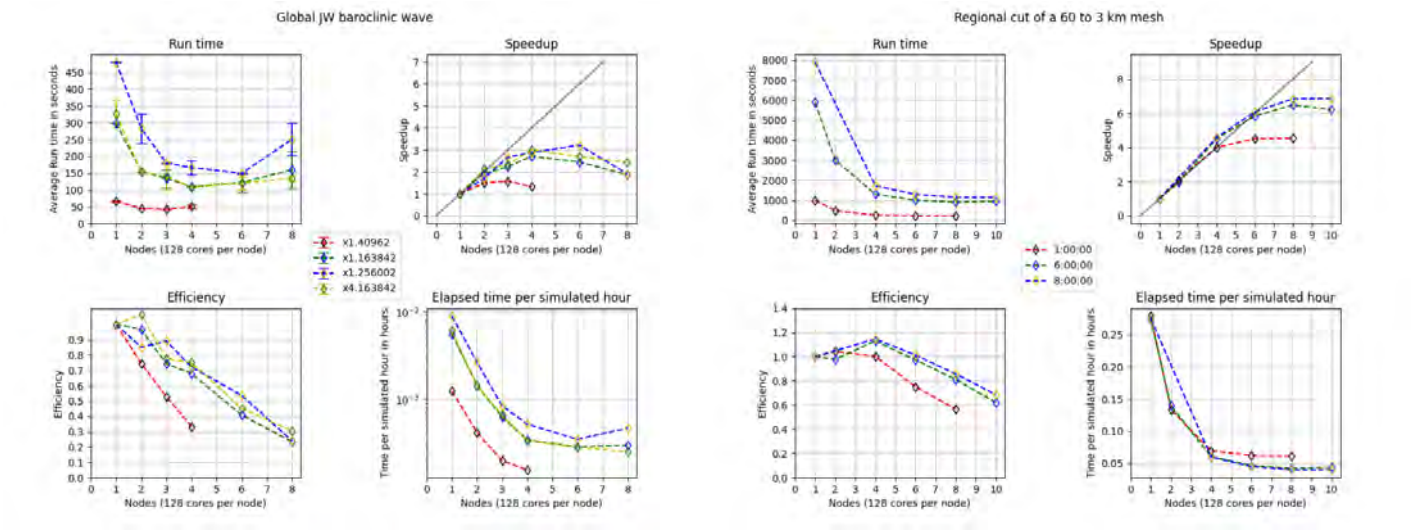


Figure 1: left: Performance of different meshes in a global 150 hour Jablonowski-Wiliamson baroclinic wave. right: Performance of different simulated times in a regional cut using a 60 to 3km mesh.

as a real case, this case allowed for good comparison between different meshes and run times without much set up in between. The first thing we investigated is how performance scales with different meshes. As MPAS uses unstructured Voronoi meshes in the model it is fairly straightforward to change between them. We used uniform meshes with a cell size of 120km (40962 total cells), 60km (163842 total cells) and 48km (256002 total cell). Note with the first two that we need four times the cells to double the resolution. With this we would expect the run time to also quadruple, as the number of cells informs how many integration steps need to be done by the model. We also tested a variable mesh using a circular refinement with cell sizes varying from 92 to 25km. This is used as it has the same number of cells as the 60km uniform mesh (163842 cells), so that we can test whether the refinement introduces any extra overhead.

Figure 1 shows the performance of the 150 hour runs. Note, that the mesh names show the refinement (x1 meaning uniform) and the second number shows the total number of cells in the mesh. The run time on a single node (128 cores each) increased by around 4.3 times from the 120km to the 60km mesh. This is slightly worse than expected, but is most likely due to 128 cores being already near the ideal for the 120km mesh, while the 60km mesh only reaches it's fastest run time on 4 nodes. The x1.163842 (60km uniform) mesh and x4.163842 (92-25km refined) mesh show very similar scaling throughout, indicating that indeed the number of cells is the important factor and refinement does not add more overhead. Overall, we found that around 320 cells per core are a good estimate for ideal performance, this number however varied slightly for each mesh. Others[2] have shown around 150 cells per core as ideal behaviour for more complex cases, so this depends on the exact use.

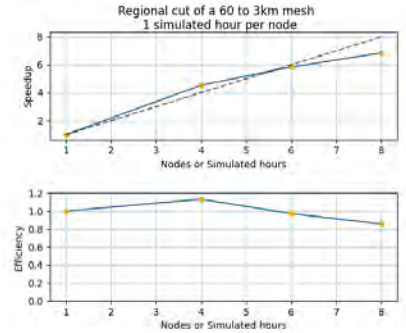


Figure 2: Weak scaling of the regional cut using 1 node per simulated hour

Regional Models

For the regional cut, we tested a simulation based on weather data from January 2010 for the Western Mediterranean. We used a mesh with varying cell sizes between 60 and 3km and simulated between 1 and 8 hours. On a single node run time increased linearly and stayed constant at about 1000s (16min 40s) per simulated hour. As seen in figure 1, there is some super-linear speedup initially. Since this does not show up in the global runs, we believe this is due to extra overhead from the model reading in updated boundary conditions every simulated hour. Good strong scaling overall is observed and the model stays over 60% efficient even after the run time plateaus, allowing us to simulate eight hours of weather in around 17 minutes. When looking at the weak scaling of the regional model (see figure 2), we observe near ideal behaviour throughout the experiment. An efficiency drop of of less than 20% at 8 nodes is promising and shows that 128 cores per simulated hour is a good estimate for efficient running of regional models.

Visualising the atmosphere

Visualisation method

Atmospheric models, like any model, are only of limited use if the outputs cannot be visualised in a way that makes the complex results easily understandable for humans. This is why we also worked on a visualisation tool for MPAS Atmosphere. Which physical parameters are being stored and at what frequency these outputs are given can be easily configured before running MPAS. The outputs are stored as compressed netCDF files, which store both the physical parameters and the mesh specifications used.

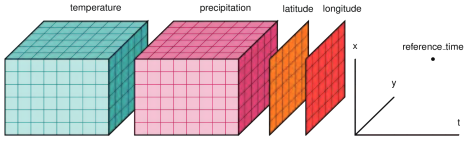


Figure 3: netCDF archive structure. Credit: Hoyer, S. & Hamman, J.. (2017). xarray: N-D labelled Arrays and data-sets in Python. Journal of Open Research Software. 5. 10.5334/jors.148.

Since they are compressed files, they save on storage but can be complicated

to work with. Luckily, Python already provides a package to interface with the netCDF library. A key point that must be considered, when working with the output from MPAS, is that there are three different types of location for each cell, a central cell location, the location of its vertices and the edge locations (see fig.:4). Furthermore there is not only a single layer of cells, but several vertical levels to deal with.

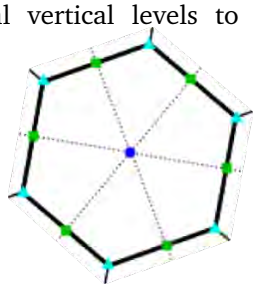


Figure 4: Stored location for a cell within the Voronoi mesh. Central cell location: dark blue, Vertex location: cyan, Edge location: light green

Which location needs to be used depends on which physical parameter is to be plotted. For example the surface pressure or the temperature are attached to the cell centre, whereas the wind speed, which can be understood as a gradient between cells is attached to the edges. There already exists a Fortran program to interpolate the MPAS output for each parameter to consistent coordinates on the longitude and latitude grid (see). After this conversion we are able to use Python to plot physical parameters in longitudes and latitudes.

The conversion step using Fortran is not necessary, but makes working with the outputs easier. In Python, standard mapping packages like Cartopy can be used to plot the data in 3D or 2D using different projections and allowing to add physical or political maps in the background to help contextualise the outputs.

Visualisation Examples

Our regional example described above outputs 38 possible physical measurements which could be plotted. The simulation started at 6 pm and ran for a total of 8 hours until 2am the next morning. Outputs were saved every hour, giving us nine time steps for plotting. We can easily create animations for the entire model with these outputs (as shown in our presentation here). Examples of these outputs are shown in Figures 5 and 6.

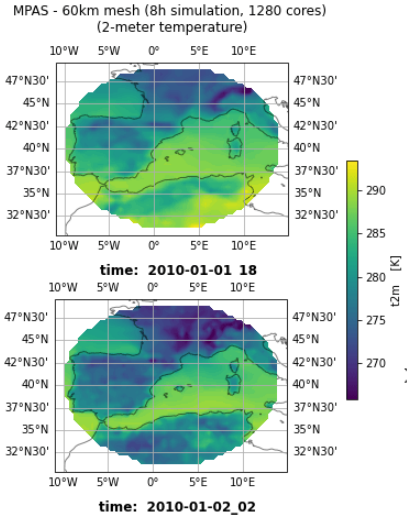


Figure 5: Surface temperature over 8 hours. An overall drop in temperatures during the night can be seen, as well as already lower temperatures in the high mountains of the Alps and Pyrenees.

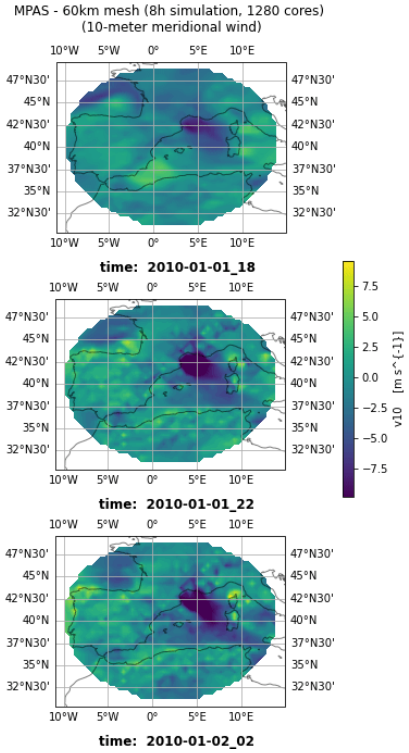


Figure 6: Meridional Wind plot over 8 hours. The dark spot in the middle of the figure, is a well know wind pattern, the so-called Mistral Wind. That this is visible in our output data, again reflexes the validity of our MPAS run.

Conclusions

MPAS Atmosphere shows good strong and weak scaling in both global and regional runs. The number of cells in the mesh and length of simulated time show the biggest impact on performance. Thanks to the flexible resolutions within models, it is possible to compensate high resolution in one are by using bigger cells away from the area of interest.

In our experiments with a regional cut covering the entire Iberian Peninsula and southern France, we were able to simulate 8 hours of weather in around 17 minutes of real time. This shows MPAS's potential to be used for numerical weather predictions.

While visualisation of the outputs requires some post processing of the netCDF outputs to get the latitude and longitudes for each data point, we developed scripts that automate this process. After this, the data can easily be plotted using Python, further easing the use of MPAS Atmosphere.

Overall, the good performance on a supercomputer like ARCHER2, the versatility of the model core and the ease of plotting the outputs makes MPAS Atmosphere promising for fast and reliable modelling of the worlds atmosphere.

References

[1] Jablonowski, C. & Williamson, D. L. (2006): A baroclinic instability test case for atmospheric model dynamical cores.

[2] Heinzeller, D. & Duda, M. G. & Kunstmann, H. (2016): Towards convection-resolving, global atmospheric simulations with the Model for Prediction Across Scales (MPAS) v3.1: an extreme scaling experiment

PRACE SoHPCProject Title

Investigating Scalability and Performance of MPAS Atmosphere Model

PRACE SoHPCSite

EPCC, Scotland

PRACE SoHPCThe authors

Jonas Eschenfelder, Imperial College London, United Kingdom Carla Nicolin Schoder, University of Vienna, Austria [association,] country

Jonas Eschenfelder

Carla Nicolin Schoder

PRACE SoHPCMentor

Evgenij Belikov, EPCC, Scotland

PRACE SoHPCCo-Mentor

Mario Antonioletti, EPCC, Scotland

PRACE SoHPCContact

Jonas Eschenfelder, Imperial College London
E-mail:
jonas.eschenfelder18@imperial.ac.uk

PRACE SoHPCSoftware applied

MPAS Atmosphere

PRACE SoHPCMore Information

http://mpas-dev.github.io/

PRACE SoHPCAcknowledgement

Special thank you to Evgenij and Mario for guiding us through this project. Thank you to the EPCC for hosting us and allowing us to work on their systems. Thank you to the SoHPC and PRACE for giving us this opportunity for a challenging but fun summer.

PRACE SoHPCProject ID

2113

Re-engineering and Optimizing the Genomicper Package

İrem Okur, Aybüke Özçelik

Current technology developments enable us to analyze more genetic data points to determine their role in disease. Project 2114 aims to increase the size of data sets that the Genomicper Package could manage and to speed up the processing of current data sets.[3]

Introduction

Developments in the science especially the field of mathematics and biology have made it possible to analyze and interpret biological data in a computer environment. This is called bioinformatics, especially when the data sets are large and complex. Bioinformatics has led to a detailed examination of the effects of genetic factors on disease susceptibility. This technology is also used in all kinds of different and wide areas, from network and system biology to drug research.

Genomic permutation or Genomicper is a R package. It is a permutation approach that uses genome-wide association studies (GWAS) results to determine the significance of gene sets and pathway associations. The GWAS study finds single nucleotide polymorphisms that are linked to trait variations.

This project includes the optimisation of the Genomicper package by using various optimisation techniques, see the methods section below..

In the study of the genomicper package, single nucleotide polymorphisms (SNPs) with trait variation are aimed to be detected by genome-wide association(GWAS).[1] Genomicper is an R package currently available from the CRAN repository, the main distribution point for R packages.

The motivation behind this project is to re-engineering and optimizing the Genomicper Package to provide the ability of to analyze huge data sets (tens of millions) by using personal computers. It can analyzes data sets ranging from 30 thousand to 8 million. The data of the project summarizes the connection between Single nucleotide polymorphism or SNPs and traits (phenotype).

Methods

Figure 1: Development cycle of the code.

The development cycle used in this project is illustrated by Figure 1. First, profiling is done and most expensive routines are identified with the help of profvis. Once an expensive routine is identified it is further profiled and performance improvements are compared against an implementation

of the original code using another R package - microbenchmarks. Then, the expectedqual function is used to check whether the operation gives correct results. If the results are the same, go to the next step, if the results are different, go back to the previous step. The final step is comparison, The processing speed of the code belonging to the previous version are compared with the code that has been optimized. If there is a decrease in processing time, the optimized code is used in place of the previous code. If the optimization process is unsuccessful, project continues with the previous code. So a cycle is completed and profiling is done to start a new cycle.

In more detail, with the help of the Profvis R package interactive graphical interface for visualizing data from package was done. When the genomicper package was profiled, the memory usage and execution time of the each routine could be easily detected. The microbenchmarks was applied to quantify improvements of changes compared to the original.

Results

Figure 2 shows the top three most expensive routines are genome_order read2_paths and snps_permutation. Optimisation of these three routines made the package more efficient. After we were satisfied with the efficiency of our code we began comparing the original version and the optimised version.

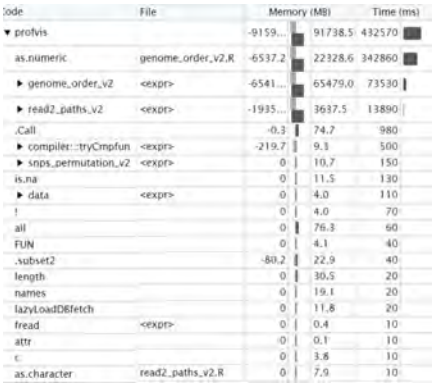


Figure 2: Profile of the package.

As we can see in Figure 3 the original code on the left and the version being optimised _v2 on the right. In _v2, we used multiple optimisation techniques. The final result which is illustrated by Figure 3, is because of Microbenchmark's multiple runs. It helped more clear to see the comparison of the original version and version two. To explain better, the fact that getting a distribution comes from this.

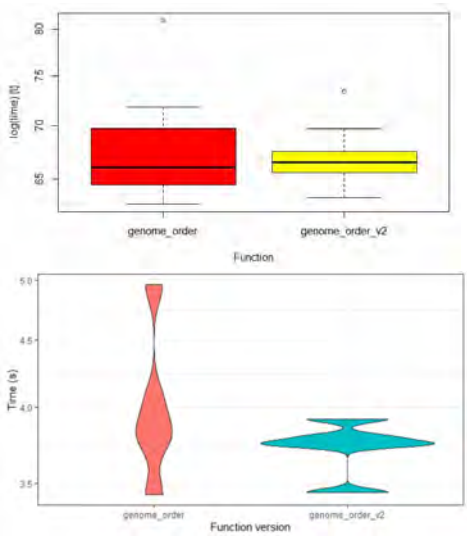


Figure 3: Comparison of the GenomeOrder routine.

Discussion and Conclusions

The project aim was to reach the larger data sets that the Genomicper Package can handle and to be able to process existing data sets faster. Optimisations that worked faster with larger data sets were preferred to those that showed performance improvements only for the smaller data sets. As a result of the improvements we have made genomicper should be able to handle larger datasets faster. More importantly, we have introduced a methodology that should continue to improve the performance of genomicper in the near future.

Future Work

The Genomicper Package is an open source project that researchers can benefit from. The project can be written in the Python programming language, because researchers who do not know the R programming language can benefit from it and because of its strong mathematical processing speed. In addition, there is a topic investigated by the team. If the data to be processed is adjusted according to the P threshold value, a significant decrease in the processing time of the R package can be expected.

References

[1] Genomic Permutation https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3429921/ [Last accessed 31 August 2021]

[2] Cran https://cran.r-project.org/ [Last accessed 31 August 2021]

[3] Cran Genomicper https://cran.r-project.org/web/packages/genomicper/genomicper.pdf [Last accessed 31 August 2021]

Acknowledgements

First, we would like to thank PRACE for giving us this opportunity. Thank you Summer of HPC team for your work in pandemic times for this online programme.
Special thanks to our project Mentor Dr. Mario Antonioletti, our Co-mentors Dr. Pau Navarro and Dr. Claudia Cabrera for their patience and interest. They helped all along the way, even it was online.

PRACE SoHPCProject Title

Re-engineering and optimizing Software for the discovery of gene sets related to disease

PRACE SoHPCSite

EPCC, UK

PRACE SoHPCAuthors

İrem Okur, Dokuz Eylül University, Turkey
Aybüke Özçelik Izmir Institute of Technology, Turkey


Irem Okur


Aybüke Ozcelik

PRACE SoHPCMentor

Dr. Mario Antonioletti, EPCC, Edinburgh

PRACE SoHPCContact

İrem Okur, Dokuz Eylül University
Phone: +90 0553 223 1362
E-mail: iremokur10@gmail.com
Aybüke Özçelik, Izmir Institute of Technology
Phone: +90 507 425 3691
E-mail: aybuke_ozcelikk@hotmail.com

PRACE SoHPCSoftware applied

R

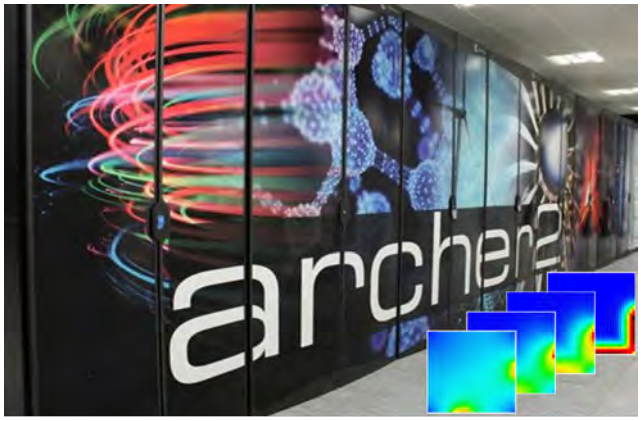
PRACE SoHPCAcknowledgement

Thanks to our mentor and co-mentors that were very helpful and understanding.

PRACE SoHPCProject ID

2114

Accelerating Python on AMD CPUs



Alejandro Dinkelberg
Jiahua Zhao

Numerical Python calculations can achieve speeds in a similar range to C and Fortran.

The broad application of Python and its simplicity lead to a huge community of users. The advantages of readability and dynamic typing for writing Python code come along with a slow computational performance which in turn makes it Python unattractive for complex calculations. Nevertheless, its popularity exerts pressure to accelerate Python.

Complex simulations and extensive numerical calculations are highly recommended candidates to execute on High Performance Computing (HPC) systems. Recently, the EPCC[1] topped up their HPC systems and is now using the UK HPC Tier-1 AMD-based Supercomputer ARCHER2. This leads to an interest of evaluating the performance of Python on AMD-based systems since there rarely exist studies on the performance improvements for AMD-based HPC systems.

The languages C and Fortran deliver the highest standards for computational performance and therefore they will play a benchmark role for our Python code. Although naive Python lags far behind, we will show here how to transform Python code to accomplish C/Fortran performance. We will mainly measure the performance on the HPC system ARCHER2 and compare it to an intel-based HPC system (Cirrus). The results will show us what improvements we can expect and how competitive Python is.

Computational Fluid Dynamics

Our performance benchmark is an example from the Computational Fluid Dynamics (CFD). It simulates a fluid flow in cavity approximating a flow pattern. The algorithm discretises the problem on a two-dimensional grid, to approximate the partial differential equations by using the Jacobi algorithm. It can be described as an iterative process which converges to a state with an approximated solution. Each iteration adds precision to the result until reaching a stable state. Additionally, increasing the grid size leads to higher precision but is also more computationally intensive.

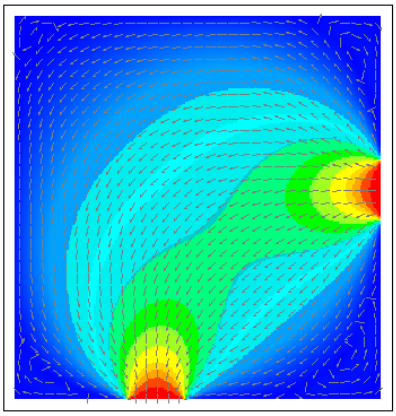


Figure 1: Example of an approximated flow pattern with in and out flow

How is the problem being solved?

The determination of the flow pattern is broken down to a finite problem size and is approximated by using a grid. For every cell , we use the stream function Ψ for zero viscosity to calculate fluid velocity:

$$\nabla^2\Psi = \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2} = 0 \qquad (1)$$

The finite version of the equation gives us the option to compute the stream by averaging the value of the cell's four nearest neighbours:

$$\begin{aligned} 0 =& \Psi_{i-1,J} + \Psi_{i+1,j} + \Psi_{L,j-1} \\ &+ \Psi_{i,j+1} - 4\Psi_{i,j} \end{aligned} \qquad (2)$$

Scalabilty of CFD approximation

1. The **Scale factor (*sf*)** determines the grid size (minimum size 32 x 32). The grid is represented as a matrix so that *sf* specifies the required memory size in our calculation.

2. The **number of iterations** is the precision parameter of the algorithm.

3. The **Reynolds number** (Re) is the degree of the fluid's viscosity. It adds complexity to the problem while making it more realistic.

Note that the main computational costs are in the iterative process of the algorithm (see Fig. 4, *jacobistepvort*). In our project, we focus on improving this process as the calculation is the same for every iteration.

Methods to improve our Python code

The benchmark for our algorithm improvements is a naively written Python program. To improve its velocity and to reduce computational costs, we introduce the following approaches:

Code optimisation - It can be obtained by vectorize the for-loops with a matrix calculation provided by the *NumPy*[2] package. Additional improvement is promised by the package *NumExpr*[3] that resolves mathematical operations efficiently to accelerate NumPy's matrix calculations.

Divide and Conquer - The serial version of our grid calculation is using just one processor at a time for the calculation. What if we divide the grid and run it on multiple processes, using multiple CPUs? The language-overarching message passing interface (MPI) is a technique for processor communication. *Mpi4py*[4] is a Python package that implements the functionality of MPI.

GPUs instead CPUs - The specialty of the GPUs is the wide-spread parallelisation of the small tasks. Therefore, GPUs often have more than hundreds of device cores. Here, we can use *Cuda* from the *Numba*[5] package to implement the code for the GPU.

Focus: Parallelisation of Python code

We use a one-dimensional domain decomposition[6] and narrow down the problem into smaller ones. In Python, we decompose our grid one-dimensional (x-dimension) to slice it into sub grids in order to allocate each sub grid to a single process. These calculations can run in parallel.

Our Jacobi algorithm is based on information about the cell's neighbours. Splitting up the grid provokes a lack of information at the boundaries of the sub grids. Hence, we have to enable the communication between the processes.

Halos are boundary layers of each sub grid that are shared with another neighbouring sub grid. If the halo of a sub grid is updated and it gets informed by its neighbouring sub grid, this sub grid updates and informs that is was updated (see Fig. 2). The halo swaps ensure that the information is up-to-date.

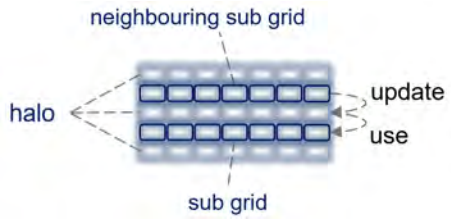


Figure 2: Sub grids performing halo swaps

About ARCHER2

EPCC's ARCHER2 is an HPE Cray EX supercomputing system with an estimated peak performance of 28 PFLOP/s. The final system will have 5,848 dual sockets compute nodes based on AMD EPYC Zen2 (Rome) 64 core CPUs running at 2.25 GHz, given 748,544 cores in total.[7] Each standard compute node has 256 GiB of DDR4 memory. Except for GPU programs (using single NVIDIA Tesla V100-SXM2-16GB on Cirrus[8]), our CPU programs run on standard compute nodes with Cray compiler (cc & ftn) and Cray python (3.8.5.0). See Fig. 3 for details.

One Standard CPU Compute Node	
Processor	2× AMD Zen2 7742, 2.25 GHz, 64-core
Memory per node	256 GiB
Interconnect	HPE Cray Slingshot, 2× 100 Gbps bi-directional per node
Operating system	HPE Cray Linux Environment (based on SLES 15)
Scheduler	Slurm
Compilers	HPE Cray Compiling Environment (CCE)
Python	Cray Python (3.8.5.0)
Parallel libraries	HPE Cray MPICH2

Figure 3: The hardware & software details of ARCHER2: standard compute node.

Profiling and optimizing

First, we run the serial Python benchmark programs using a single rank. Given the problem size $sf = 256$, the naive Python code needs about 896 sec per iteration, whereas the NumPy version only takes about 1.2 sec. We use cProfile to analyze the time consumption during run time (see Fig. 4).

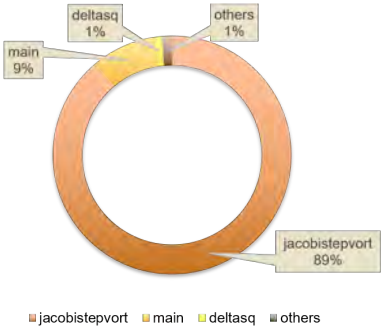


Figure 4: The hotspot functions in Python programs.

The function *jacobistepvort* is computationally intense and takes up to 90% of the time, followed by *main* and *deltasq*. This loop implements the equations for finite viscosity which are a much more complicated than Equation (2) but have the same general structure where each point depends on its four neighbours. Further on, we focus on *jacobistepvort*. The naive Python code updates the data at each point by using double for-loops, whereas the NumPy version allocates the memory before calculating and iterates internally to update the data, which is more efficiently implemented. To speed up the Numpy version, we introduce NumExpr to speed up the array calculation. We use the function *evaluate* to convert the original calculation expression into a fast numeric expression and to optimize the data access. In addition, we speed up naive Python by using Numba shifting the *jacobistepvort* calculation to the GPU platform.

Results for serial versions

The serial Python programs, except for Numba version running on a GPU on Cirrus, were tested on ARCHER2 (see Fig. 5).

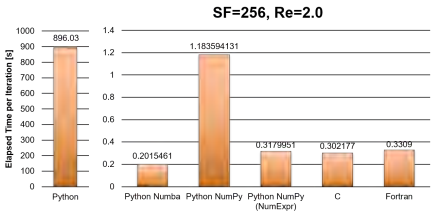


Figure 5: Performance: serial (single CPU core / GPU).

In comparison to the serial versions of the CFD program, the implementation of the GPU-based version is 6 times faster than the NumPy version and even slightly faster than the C and Fortran versions. It is important to ensure that the data copying from CPU to GPU and back is minimised. The NumPy version is nearly 800 times faster than naive Python, and the performance of NumPy version after NumExpr optimization improves greatly, even compared to the performance of C and Fortran versions. This indicates that NumExpr has a good optimization for NumPy arrays calculation at this problem size.

Parallel programming

To improve the comparability and the evaluation of the performance, we use the serial NumPy version as a baseline

(1.18s per iteration). In Fig. 6 we display the results of the parallel versions on one ARCHER2 node for: C-MPI, Fortran MPI and Python with NumPy MPI and with NumPy using NumExpr.

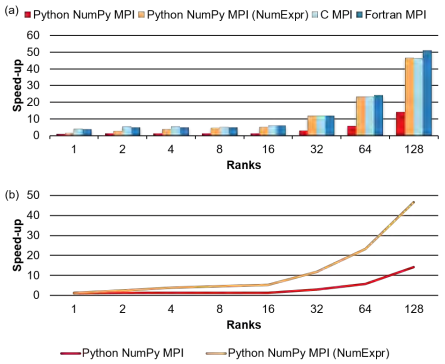


Figure 6: Performance: parallel (single node / multiple ranks).

Overall, the speed-up of all parallel programs increases with the number of ranks (see Fig. 6(a)). For the Python programs, the performance improvement is small with a small number of ranks (1 to 16), as well as for C and Fortran. The NumExpr optimized version is close to C and Fortran at 16 ranks. In the case of a large number of ranks (32 to 128), the performance of all programs improves significantly. The Python NumPy MPI consistently lags behind the other three, while the performance of NumExpr optimized version is still close to C and Fortran. In Fig. 6(b) we see that after more than 16 ranks, the scalability of the NumExpr version is better than non-NumExpr version.

When the number of ranks (multiple compute nodes) increases further (Fig. 7), the scalability of Python programs decreases, especially after 256 ranks. The performance of C and Fortran programs increases sharply, while the speed-up of Python programs increases slowly (see Fig. 7(a)). In particular, it is shown that the scalability of non-NumExpr version is better than NumExpr in Fig. 7(b). It is slightly better than linear while going from 128 to 2048 ranks. The C and Fortran scalability is massively super-linear, presumably because of cache effects: as the local problem size gets smaller, it suddenly fits into cache and gets a huge performance jump (between 256 and 512 ranks). Mpi4py may have more serious blocking problems because the communication overhead across nodes is often larger than that within nodes. We would like to address this problem in future work as we expect also a speed-up with mpi4py on more ranks.

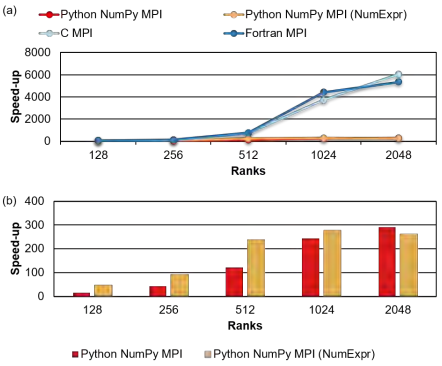


Figure 7: Performance: parallel (multi-node).

Conclusion

Our work demonstrated the feasibility of Python HPC, and we are convinced that Python solves scientific problems in a flexible manner. Python can be used efficiently on HPC systems when combined with scientific acceleration packages such as NumPy, NumExpr, Numba and so on. For allocating the problem over multiple nodes and achieve high-speed parallel computing, Mpi4py can be used. Our results show that the combination of NumPy and NumExpr gives Python programs similar performance to C and Fortran in the serial case. In the parallel case, if the number of cores is at its maximum and there is no cross-node communication, the performance of Python programs (about 48x speed-up) combined with "*NumPy + NumExpr + Mpi4py*" is close to C and Fortran.

Nevertheless, if multi-nodes are used, the NumExpr Python version has no advantage here and the Numpy version is more extensible. Of course, C and Fortran still have unbeatable computational advantages for large-scale computing, so we recommend that researchers choose the right programming language for the actual problem: If you want to solve the problem more convenient, you can use Python although you should use accelerating techniques. Naive Python is not recommended at all. If you want to solve the problem on large-scale nodes, you may prefer C or Fortran with MPI to receive the most efficient results.

Future work

We will explore Python for large-scale computing in the future. It will be interesting to investigate larger problem sizes and optimize Python performance on more nodes on ARCHER2, and we will try to optimize it more for AMD Zen2 architecture.

At present, we are only beginning to use the Heterogeneous Computing Platforms (GPUs) to accelerate Python programs, and we have not yet fully utilized the computing power of GPUs. So in the future, we will improve GPU acceleration performance for Python programs and implement Python programs which support multi-GPU use.

Acknowledgement

Thanks for a fantastic virtual Summer of HPC, we have learned Python HPC skills, which will be of great help to our future. Thanks to our mentors David Henty and Stephen Farr, to the Summer of HPC organisers and to friendly partners.

References

[1] EPCC at The University of Edinburgh. https://www.epcc.ed.ac.uk/

[2] NumPy. https://numpy.org/

[3] NumExpr Documentation Reference. https://numexpr.readthedocs.io/projects/NumExpr3/en/latest/

[4] MPI for Python. https://mpi4py.readthedocs.io/en/stable/

[5] Numba: A High Performance Python Compiler. numba.pydata.org/

[6] Gropp, W. D., & Keyes, D. E. (1992). Domain decomposition methods in computational fluid dynamics.

[7] ARCHER2 Hardware & Software. https://www.archer2.ac.uk/about/hardware.html

[8] Cirrus powered by EPCC. https://www.cirrus.ac.uk/

PRACE SoHPCProject Title

Performance of Parallel Python Programs on ARCHER2

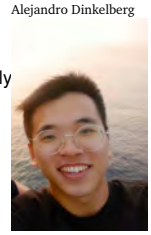
PRACE SoHPCSite

EPCC, UK

PRACE SoHPCAuthors

Alejandro Dinkelberg, University of Limerick, Ireland
Jiahua Zhao, University of Padua, Italy


Alejandro Dinkelberg


Jiahua Zhao

PRACE SoHPCMentor

Dr. David Henty, EPCC, UK

PRACE SoHPCContact

Alejandro, Dinkelberg, University of Limerick
E-mail: Alejandro.Dinkelberg@ul.ie
Jiahua, Zhao, University of Padua
E-mail: jiahua.zhao@studenti.unipd.it

PRACE SoHPCSoftware applied

C, Fortran, Python, cProfile, NumPy, NumExpr, Numba, Mpi4py

PRACE SoHPCMore Information

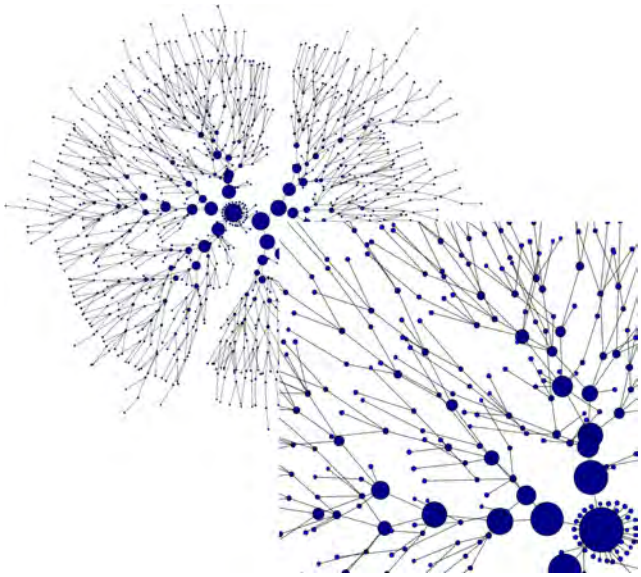
https://summerofhpc.prace-ri.eu/performance-of-parallel-python-programs-on-archer2/

PRACE SoHPCProject ID

2115

Branch Bound for SoHPC

Valentin Trophime
& Oliver Legg



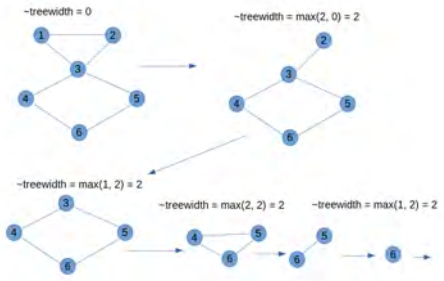
The goal of the project is to accelerate the computation of tensor network contractions, which are used to simulate quantum computers. Parallelising tree-width computation and clever heuristics can help achieve these goals.

Our project is motivated by the need of simulating a quantum computer. In order to evaluate the performances of real quantum computers we need to compare them to “what we expected”. This expectation is the result of the simulation of the virtual quantum computer inside a classic computer. Simulating such a computer is done by contracting a tensor network this is basically a great multiplication of tensors of different dimensions. There are many possible orderings, (choose where you put the parenthesis in your expression) to compute this multiplication - some are more efficient than others. A dual problem of this is to find the best vertex elimination order that gives the best tree-width of the network in the end of an elimination process.

Finding the best order is known as an NP problem therefore the current approaches in literature are based on heuristics that give an approximation or on "clever brute-force" techniques. Our goal is to implement two algorithms, one based on heuristics and flows called Flow Cutter and another using branch and bound paradigm with parallelism called FPBB. For fast parallel branch and bound, Valentin implemented the flow cutter algorithm in a Julia package and Oliver implemented the QuickBB algorithm. An explanation will be provided for how flow-cutter works. However, the original paper[1] is available if necessary. Likewise, the QuickBB algorithm will have an explanation and the paper[2] will be available.

Vertex Elimination

The first notion necessary to dive into our problem is the vertex elimination. It is a simple procedure which requires a graph and a sequence of nodes. This sequence is called a vertex elimination order because it will tell us when a node will be removed in the procedure. The procedure is basically taking a node from the sequence, connecting all of its neighbours together then removing it, and repeating the cycle. While doing so we manage a variable named tree-width equals to the maximum degree all nodes during this process. Here is a visual example of this process on a simple graph:

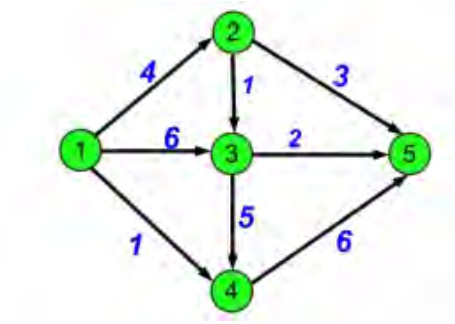


Flow

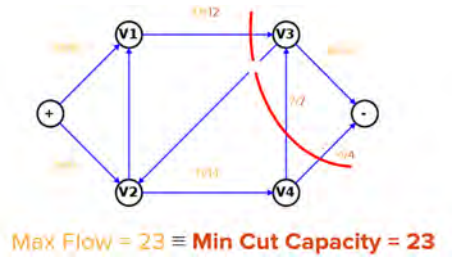
A flow is an integer value between 0 and C where C is the capacity of the pipe (i.e the maximum amount of water/info/whatever you can put in it). We also introduce 2 notions for nodes, source and target. A source node is essentially a node that creates flows so that it can throw more than it receives. The opposite is the target node which absorbs flows so it can receive more than it gives. The other nodes respect the flow conservation i.e same amount

of flow in and out.

Here is an example of a flow graph with 1 as the source and 5 as a target, the current flow is $11(=4+6+1)$ and in other nodes we can see the conservation law for instance in node 2 we have 4 in and 3 + 1 out.



Finding the best way to flood a network to transport things from a source node to a target node is a well-known problem and it is demonstrated that finding the maximum flow is equivalent to find a minimal capacity cut. It will primarily try to solve this max flow problem to find cuts.



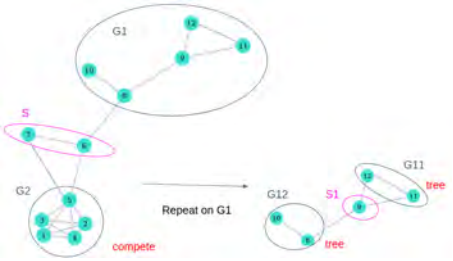
Cut and separator

A cut is a partition in two disjoint sets $V_1 \cup V_2 = V$ of all the nodes in a graph. We also refer a cut as a set of arcs/edges with one side in V_1 and the other side in V_2. The size of a cut is defined as the number of cut-arcs in it, and its expansion as $\frac{size}{min(|V_1|,|V_2|)}$. We can also define a imbalance rate $\epsilon = \frac{2 \cdot max(|V_1|,|V_2|)}{n} - 1$ where $n = |V|$. Knowing those 2 numbers, we can say that a cut C_1 is dominated by another cut C_2 if and only if $C_1.size \geq C_2.size \wedge C_1.\epsilon \geq C_2.\epsilon$. From a cut we can compute a separator.

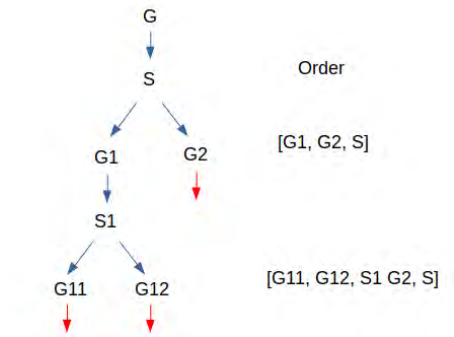
A separator is similar to a cut but it is a partition in three disjoint sets $V_1 \cup Q \cup V_2 = V$ and there is no edges/cuts between $V1$ and $V2$. Separator also has size $(=|Q|)$, expansion and imbalance rate (same definition respectively as for cut), therefore separator can dominate each other too. The way Hamann and Strasser suggest to do the conversion cut to separator is by picking one node from the largest side for every cut-arc and put it in Q. Their objective is to use separator to split graph in two balanced halves.

Nested dissection

This algorithm is made of three layers: the nested dissection, the separator and the core flow-cutter.



Nested dissection consists of taking a graph, computing a separator, with which you cut the graph in 2 parts (or more) i.e 2 disjointed sub graphs. Repeat the same process on the 2 sub graphs until they verify some properties like “is a tree” or “is complete” where it’s easy to compute their order and tree-width. To create the final order for the root graph, start from the end by adding separators then graph orders (when they verify the previous properties and are computed directly). For example, if you had a graph G, its separator is S and it splits G in G1 and G2, at this point the order is [order(G1), order(G2), S] (I write S but in reality we append all of its nodes and order doesn’t matter inside S, the only important thing is that S is after G1 and G2). Now let us start to nest things with G1 and suppose that G2 can’t continue because it is complete. G1 will be split using S1 and its sub graphs are G11 and G12. At this point the order is the following: [order(G11),order(G12), S1,order(G2), S]. For the sake of simplicity, we will suppose that G11 and G12 are also complete to end the example here. Just one note here, our order [order(G11),order(G12), S1,order(G2), S] and this one [order(G11),order(G12),order(G2), S1 S] lead to the same tree-width so picking one or another depends on your choice when thinking about implementation. Another way to think about this approach is more visual and involve a tree like this. Therefore, the choice depends on how you want to iterate through this tree.



For performances reasons we will try an iterative version even if recursion seems nice here. During the main loop we try to maintain the count of tree-width by counting cell’s size with one formula from this other paper[3] from the same authors. Cell here is a data structure define in the paper, it has two sets of nodes: interior and boundary. Its size is define as $|interior| + |boundary|$. Knowing that we can condense the dissection layer in this pseudo code:

```
Function dissection(graph) {
    order = []
    tw = 0
    queue = Queue<Cell>()
    queue.enqueue(Cell(interior = graph.nodes, boundary = None))
    while queue is not empty {
        cell = queue.dequeue()
        subgraph = graph.getSubgraphOf(cell.interior)
        if g is complete {
            order.appendAtStart(subgraph.nodes)
            tw = max(tw, cell.size – 1)
        }
        elseif g is tree {
            # get optimal order for tree: leafs to root
            order.appendAtStart(getTreeOrder(subgraph))
            tw = max(tw, length(cell.boundary) + 1)
        }
        else {
            # compute a separator and the 2 disjoint new subgraphs
            sep, newInteriors = getSeparator(subgraph)
            tw = max(tw, sep.size + length(cell.boundary) - 1)
            order.appendAtStart(sep.nodes)
            newBoundary = union(sep, cell.boundary)
            # new boundaries = subset of cell.boundary U sep.nodes with
            # all nodes neighbors of new interior
            newCells = newInteriors.toCells(graph)
            queue.enqueueAll(newCells)
        }
    }
    return (order, tw)
}
```

Separator from flow-cutter

This layer simply calls the flow-cutter core algorithm with random inputs several times, and collect all those cuts in a collection. In practice we use multi-threading with the Threads.@threads macro from Julia to parallelise this part. After what we removed dominated cuts and select the one with min expansion before converting it to separator and returning it. This can be write in this pseudo code:

```
Function getSeparator(subgraph, nsample=20) {
    cuts = []
    for i in 1..nsample {
        # get 2 distinct random nodes
        s, t = subgraph.chooseRandomNodes(2)
        cuts.append(flowcutter(subgraph, s, t))
    }
    cuts.removeDominated()
    selectedCut = cuts.minBy(cut → cut.expansion)
    sep = selectedCut.arcs.map((v1, v2) → chooseOneFromLargerSide(subgraph, v1, v2))
    return sep
}
```

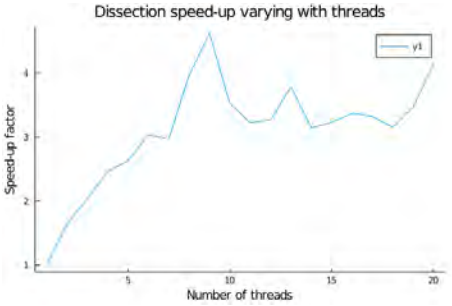
Flow-Cutter core

Now it's time to explore how the flow-cutter algorithm works and how it chooses cuts. We will use the idea of flows on undirected graphs so we double each edge to 2 arcs in both directions. All the capacities will be 1 because we don't have capacities in the input graph we use and also because it has a clever simplification with residual graphs that we no more need to compute. The algorithm will manage 4 sets of nodes S, T, SR and TR which mean sources, target, source-reachable and target-reachable. The idea is to solve the max flow problem, find the min cut and then add source (if source reachable size is too small) or target (if target reachable size is too small) in order to enhance the cut's balance by resolving the max flow problem and repeating the process several times to get more and more balanced cuts. The pseudo code given in the paper is the following:

```
1  S ← {s}; T ← {t};
2  S_R ← S; T_R ← T;
3  forward-grow S_R; backward-grow T_R;
4  while S ∩ T = ∅ do
5      if S_R ∩ T_R ≠ ∅ then
6          augment flow by one;
7          S_R ← S; T_R ← T;
8          forward-grow S_R; backward-grow T_R;
9      else
10         if |S_R| ≤ |T_R| then
11             forward-grow S;
               // now S = S_R
12             output source side cut edges;
13             x ← pierce node;
14             S ← S ∪ {x}; S_R ← S_R ∪ {x};
15             forward-grow S_R;
16         else
               // Analogous for target side
17         end
18     end
19 end
```

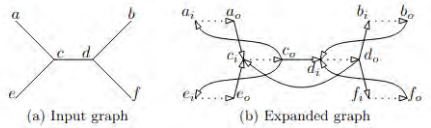
Practical results

Here is a speed-up graph obtained by applying multiple threads on the whole algorithm to the **Sycamore 53 20 graph** ($\approx$ 2000 nodes and $\approx$ 4000 edges) on one node of Kay, a super computer from ICHEC, with different number of threads. I measured the average execution time on 20 calls for each number of threads.



As the single threaded version take around 4.5 sec on Kay, the optimal 9-threads version should be around 1 sec. Therefore the performances of the algorithm are good but not as good as expected. Furthermore comparing our implementation to the one provided by the authors in C++, our gives different results of tree-width way to high.

We also implemented an MPI wrapper for both algorithms to compare with the authors code. Concerning flow-cutter, the wrapper will start different processes running the algorithm in a loop for a given amount of time. Each call has a different seed and each process gets the best tree-width and the best-order we find. In the end processes compare their results and select the best pair of orders/tree-widths. On the **Sycamore 53 20 graph**, according to the the implementation provided by the authors of the paper, we should get a tree-width around 57 for 30s of computation. Within the same amount of time, (8 processes, 9 threads on 2 nodes of Kay) we reach 94, which is quite far from the expected value. Waiting longer doesn't seem to help a lot, for 2min so 4 times longer we get 92 so a minor 5%($=\frac{(94-92)}{(94-57)}$) improvement. We suspect this to happen because we choose to keep an undirected graph in the core flow-cutter part and not to use the expanded graph.



(a) Input graph (b) Expanded graph

The expanded graph is a used to map an undirected graph to a directed one. If we use an expanded graph, the forward grow and backward grow will be different – which is why it leads to our unexpected behaviour. We suspect the issue to be related to the computation of the the tree-width during the main loop in the dissection. We encountered problems in the implementation regarding values returning with a distance of 1 to the value expected by a posterior computing the tree-width from the order. This difference is mainly due to a formula used to compute size of close cells. This is detailed in this paper,[3] which seems true but actually is false in some specific counter examples. Strasser states, the formula is incorrect only when a sub-optimal separator is chosen. A way to fix this is to create an expanded graph before calling flow-cutter and change the growing method to differentiate forward and backward.

Conclusion

I conclusion, this project ended with mixed results. In terms of pure performances and code quality, we think the project has been very enjoyable. However, it was disappointing to not have enough time to find the lowest tree-width. SoHPC was an amazing experience where we learned a lot about MPI, multi-threading, Julia and improved our English - we would love to be a part of it again.

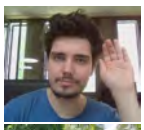
[1] Michael Hamann, Ben Strasser Graph Bisection with Pareto-Optimization *Institute of Theoretical Informatics*, Karlsruhe Institute of Technology

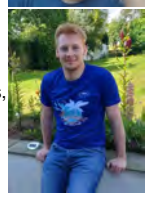
[2] Vibhav Gogate and Rina Dechter A Complete Anytime Algorithm for Treewidth,

[3] Michael Hamann, Ben Strasser Computing Tree Decompositions with FlowCutter: PACE 2017 Submission *Institute of Theoretical Informatics*, Karlsruhe Institute of Technology

PRACE SoHPCProject Title

Parallel anytime branch and bound algorithms for finding the treewidth of graphs





Valentin Trophime & Oliver Legg photo

PRACE SoHPCSite

ICHEC (Irish Center for High End Computing), Ireland

PRACE SoHPCAuthors

Valentin Trophime, Télécom SudParis, France
Oliver Legg, United Kingdom

PRACE SoHPCMentor

John Brennan, ICHEC, Ireland Niall Moran, ICHEC, Ireland

PRACE SoHPCContact

Valentin, Trophime, Télécom SudParis
Phone: +33 6 58 74 15 92
E-mail: valentin.trophime@telecom-sudparis.eu
Oliver, Legg, United Kingdom
E-mail: oliglegg@hotmail.co.uk

PRACE SoHPCSoftware applied

Julia, VertexEliminationOrder

PRACE SoHPCMore Information

julialang.org Github repo

PRACE SoHPCAcknowledgement

We'd like to express our gratitude to Simon Wong, Chris Werner, Leon Kos, and everyone else who helped organise this Summer of HPC. Also, many thanks to John Brennan for his patience in answering our questions and for his assistance in reviewing our code.

PRACE SoHPCProject ID

2116

Working with geospatial data on HPC

Rabia Özdoğan

Earth observation data can include many layers, which refer to resolutions. High spatial, spectral, and temporal resolutions have significantly increased data volumes, and analyzing the data requires high-performance computing.

We know the effects of the industrial revolution and the importance of the information it provides us. So much so that we even determined the historical periods of this revolution. What would the HPC implementation era look like if we divided the importance of geospatial data and how it changes the world with the information it provides?

When this implementation era brings together applications such as artificial intelligence and machine learning, how do we analyze the world's information?

According to European Space Agency (ESA) annual report 2020, in the Sentinel family, a total of 50,964,670 products had been downloaded by users since the start of data access operations, with a total data volume of 41.35 PB. Over 500 EO satellites have been launched in the last half-century. With the metadata provided by these satellites, we can instantly observe the changes in the world.

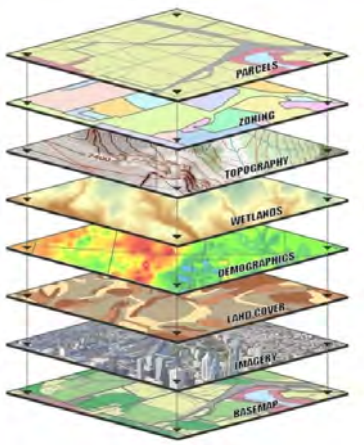


Figure 1: Layers of the geospatial data

Working with GIS data brings metadata even with the specific time and place. Handling this Metadata requires the processing of multiple stages. MPI is designed to allow the execution of multiple processes on different processors. Multiple processes work concurrently to communicate and collaborate with each other. There are many satellites for working earth observation. In our case, we use sentinel 2 which provide us optical observation about both land and water.

Workflow

Before working with GIS data, we learned to process the data we obtained as open source from Esa (European space agency). This preliminary step was done with Snap software and allowed us to obtain your XML file. We tried to get this file again while working in the terminal of the supercomputers provided by The Irish Center for High-End Computing (ICHEC). Briefly, the process was repeated in two stages as usual computing and high-performance computing.

- Pre-processing;
  *Spectral selection,
  *Spatial selection
- Processing with Graph Processing Tool (GPT)
- Installing Snap Software and GPT on HPC.

- **working in the terminal of the supercomputers provided by The Irish Center for High-End Computing**.
- Using XML file while working on HPC.

The process

We determined our work area with the Spatial Resolution and Spectral Bands selections that we applied to this metadata.Even this process can take time on our normal computers.

We can do these operations via command Shell or the Graph Processing Tool (GPT) with python that we can run jointly on Windows or Unix systems. So we can do these processes using gpt and python,with the HPC provided by The Irish Center for High-End Computing (ICHEC) at NUI Galway, without going into snap software of our computers.

Although we work in a specific region when dealing with GIS data, more than one layer is used in studies. If there is a layer, spatial or non-spatial machine learning can be applied to it. But in the analysis of GIS data for the multiple layers used in this process in our normal computer, it lasts too long for a very small region.

So with XML file of that metadata, we can do these steps on the High-Performance Computing (HPC) terminal. With this implementation, the processing of the metadata is getting significantly faster.

We can obtain the subgroup we created from these by entering the specific algorithmic calculation. Necessary corrections can be made for the analysis of the image. In our case, clear cloudless data was required for the detection of algae, so we applied also atmospheric correction from the beginning.

Figure 2: After first correction

Finding true keys

All layers were combined in a subgroup to refer to a certain height. Necessary calculations in our case ndvi index and channel selections are used in determining the algal region were made.
Corrections are like finding true keys that belong to different doors. The selected study area was Malahide which is situated approximately 18 km northeast of Dublin city. There are many corrections and calculations of the band selection.
First, the atmospheric correction was needed for a clear vision of the area. Then two necessary bands, red (B4), and near-infrared (B8) were selected.

$$NDVI = \frac{NIR - RED}{NIR + RED} \qquad NDVI = \frac{B8 - B4}{B8 + B4}$$

Figure 3: Equation of the normalized difference vegetation index

Because near-infrared light is strongly reflected, and red light is absorbed by vegetation; the vegetation index is good for quantifying the amount of vegetation(Algal area) in the sea.[1]

Bil	Vegetation Classes	Description	NDVI Value
1	Non-Vegetation	Barren areas, build up area, road network	: -1 to 0.199
2	Low Vegetation	Shrub and grassland	: 0.2 to 0.5
3	High Vegetation	Temperate and Tropical urban forest	: 0.501 to 1.0

Figure 4: vegetation classes and NDVI value comparison

Figure 5: after NDVI calculations

Figure 6: Before last step

More hands to find true keys

This work can be thought of as a key basket for many doors. Through the literature search, we learned which doors to pass through, but what about the keys?

It can be thought of as the part where the necessary calculations are made by selecting a specific region and properties of the metadata and mapping them to the keys. Using High-performance computing, we still have to calculate the pairing phase of these keys. Again, there may be human-induced calculation errors. However, the stage of finding the key made the process of each calculation stage is faster, brings us to the last door a lot faster.

Acknowledgements

Special thanks to Sita Karki who opens to new doors that I didn't know.

Award statement

I don't think that I deserve a reward for blog posting and doing other jobs aside from searching related articles and implementation. This particular area taught me that work hard and try to explore more about an inspirational thing. When I learned more things, my mind occupied more with that. I always wondered that when I reach my reach my really loved topic, how it would change me. I want to thank you all for your understanding and making me realize myself more)

Reference

[1] Hashim, H., Abd Latif, Z., and Adnan, N. A.: URBAN VEGETATION CLASSIFICATION WITH NDVI THRESHOLD VALUE METHOD WITH VERY HIGH RESOLUTION (VHR) PLEIADES IMAGERY, Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLII-4/W16, 237–240, https://doi.org/10.5194/isprs-archives-XLII-4-W16-237-2019, 2019.

PRACE SoHPCProject Title
Parallelizing Earth Observation Workflow

PRACE SoHPCSite
Irish Center for High-End Computing(ICHEC), Galway,Irland

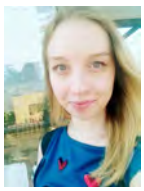
PRACE SoHPCAuthor
Rabia Özdoğan, Izmir Institute of Technology, Turkey

PRACE SoHPCMentor
Sita Karki, ICHEC, Ireland

PRACE SoHPCSoftware applied
Sentinel Application Platform (SNAP)

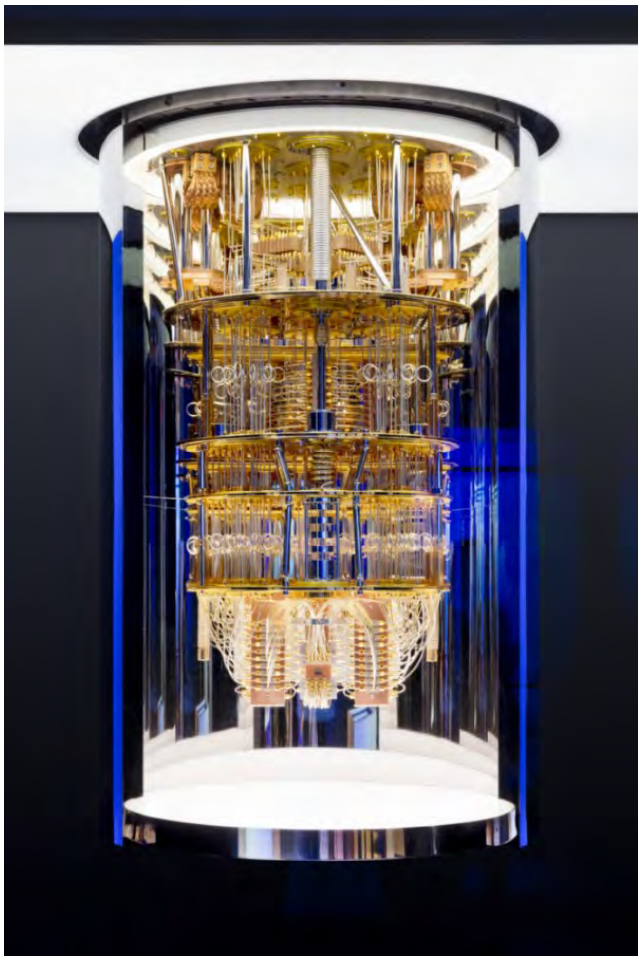
PRACE SoHPCMore Information
http://https://step.esa.int

PRACE SoHPCProject ID
2117



Rabia Özdogan

Molecular Dynamics on Quantum Computers



By Carola Ciaramelletti and Jenay Patel

The aim of this project is to find the ground state energy of a hydrogen system, using a variational quantum eigensolver (VQE) together with the quasi-classical trajectories (QCT) method, to simulate molecular dynamics. These quantum-classical hybrid algorithms are used to solve for the minimum eigenvalues of the Hamiltonian matrix, in the Schrödinger equation. The end product is a molecular dynamic simulation of the hydrogen molecule, written in IBM's Qiskit.

Quantum computing is one of today's scientific hot topics, with the potential to vastly improve our computation capacity in certain applications, due to its use of quantum-bits, offering superpositions and much faster computations. According to IBM, "quantum computing harnesses the phenomena of quantum mechanics, to deliver a huge leap forward in computation, to solve certain problems." IBM is a world leader in the quantum field, and have developed an open-source software called Qiskit, which we have used in the following research, to work with IBM Q quantum processors.

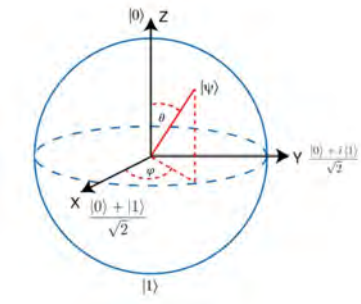


Figure 1: Bloch Sphere.

One of the most promising applications of quantum computing is used in chemistry, to solve classically intractable electronic structure problems. This in turn helps us to understand reaction rates better, and thus provides scope for drug discovery. It means that we can make predictions in quantum chemistry, without physically going into the lab and testing out different reactions. However, these simulations become increasingly more difficult as the molecules get larger and more complex. Right now, we are at the infancy stage of quantum computing, so you can only imag-

ine the ground-breaking discoveries we will make when its full potential is unlocked.

Overview

As mentioned, the project is split into two main parts: VQE, and the Quasi-classical trajectories method (QCT). The aim is to join these two parts into one molecular dynamics simulation to get an accurate solution to the Hamiltonian.

The Hamiltonian acts on a wave function, and can be difficult to solve as it requires excessive computation. However, using the Born-Oppenheimer approximation, we can split the Hamiltonian into two parts and solve them separately. The separation is done based on the assumption that the nuclei is much heavier in mass than electrons therefore, the position of the nuclei is assumed fixed as they do not feel repulsion or attraction forces as much. In turn, when representing the quantum mechanical energy of the system, the nuclear kinetic energy term can be neglected. However, the approximation still takes into account the changes in position of the nuclei to determine the electronic energy. Solving the Schrödinger equation for the electronic part, leads to the rotational dynamics of the molecule, while solving the nuclear part leads to the vibration of the molecule. Together, this gives us a molecular rotovibrational spectrum for the hydrogen molecule we are modelling, shown by the dynamic simulations.

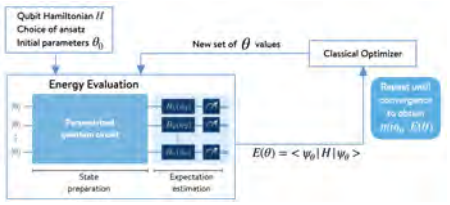


Figure 2: VQE algorithm visual schematic.

The VQE algorithm works by performing energy measurements, using quantum simulators or real quantum hardware. These energy values are the sum of the measurements of the expectation values, of each term that contributes to the Hamiltonian. The computed energy is fine-tuned using a classical optimiser (either COBYLA or SLSQP), to obtain a new set of variational parameters. These are then used to prepare a new state on the simulator or hardware. This process repeats until convergence. In our situation, we use a general approach called *the quasi-classical trajectories method* (QCT), which combines the VQE with the equations of motion

$$\begin{cases} \frac{dR_k}{dt} = \frac{P_k}{M_k}; \\ \frac{dP_k}{dt} = \vec{F}_k \end{cases}$$

with

$$\vec{F}_k = -\nabla_k E_\alpha(R)$$

and

$$E(R) \Longrightarrow \text{VQE energy},$$

in order to study molecular dynamics.

Methods

In our work, two different environments were compared for the calculation of the VQE curve, such as statevector simulator, which represents the ideal curve, and qasm simulator, as well as two different optimizers, such as COBYLA (*constrained optimization by linear approximation*) and SLSQP (*sequential least squares programming*), and two different variational forms for the ansatz: `UCCSD`, that is *Unitary Coupled-Cluster Single and Double Excitation*, and `EfficientSU2`. UCCSD is constructed by applying a single and double electron excitation operator exponentiated to an initial state, commonly chosen to be the Hartree Fock mean field wave function, while EfficientSU2 is a circuit consists of layers of single qubit operations spanned by SU(2) and CX entanglements.

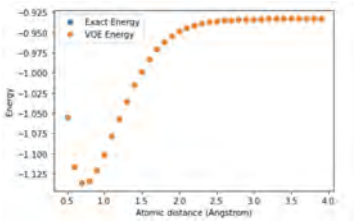


Figure 3: statevector simulator.

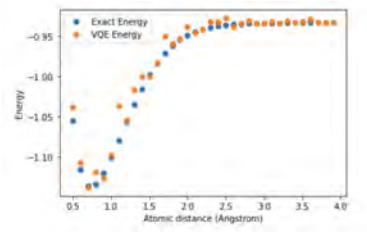


Figure 4: qasm simulator, COBYLA optimizer, UCCSD var form.

Moreover, the run was performed on two different processors, such as Falcon and Canary, to which the ibmq_lima and ibmq_armonk devices correspond. The comparison between the two different devices is respect to the ideal model. Were compared the curves obtained with the noise-model referred to the processors mentioned, and with the noise-mitigations technique. It was also chosen to compare the two different optimizers for each of the variational forms for the run done on the real quantum computer.
Ultimately, it was chosen to compare the various techniques with a single device, which is ibmq_armonk, choosing UCCSD as variational form and SPSA as the optimizer, that is a descent method capable of finding global minima, sharing this property with other methods as simulated annealing. Its main feature is the gradient approximation, which requires only two measurements of the objective function, regardless of the dimension of the optimization problem. It can be used in the presence of noise, and it is therefore indicated in situations involving measurement uncertainty on a quantum computation when finding a minimum.[1]

Molecular dynamics simulations were obtained by solving the system of equations of motion with the iterative calculation of the VQE potential, through the Runge-Kutta method. Runs were performed using the environment statevector simulator and with UCCSD as variational form and SPSA as optimizer, which proved useful for converging with less than 200 iterations.
The comparison was made for different initial conditions, where both the vibrations (in which both atoms posses zero momentum) and rotovibrations (which include momentum of both atoms) of the molecule can be observed.

Results

Below are shown the graphs of the plots of the various VQE comparisons, and some images taken from a rotovibrational simulation.

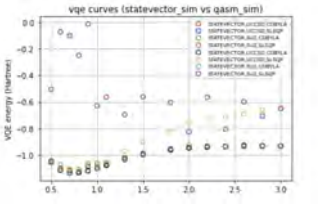


Figure 5: statevector vs qasm simulator.

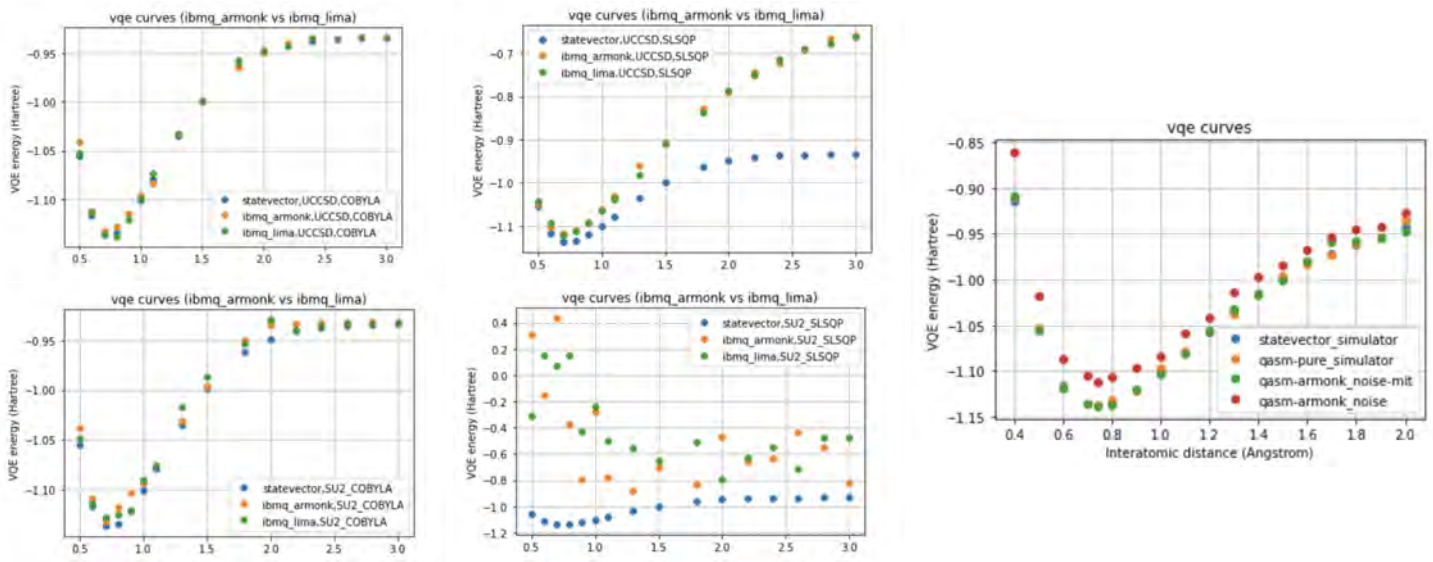


Figure 6: Comparison between Falcon and Canary processors performed respect to the ideal model, where UCCSD variational form is on top, and SU2 variational form is below. The graph on the right shows the run only on Canary, with SLSQP optimizer and UCCSD as variational form, showing the data acquired with and without noise and noise mitigation.

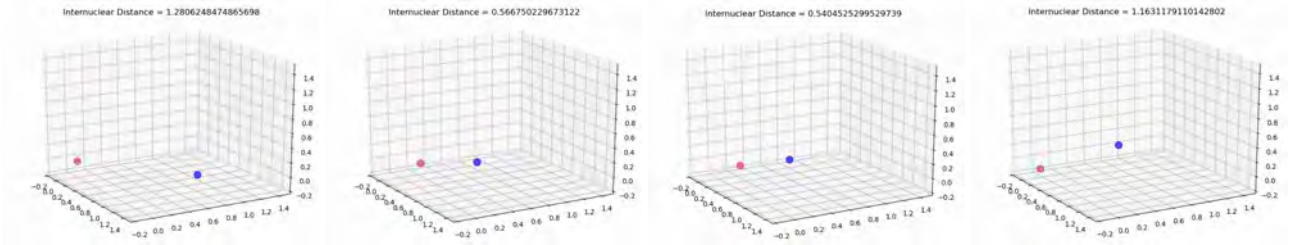


Figure 7: Some images of the dynamics simulation of H_2, calculated with the following initial conditions: $r1 = (0, 0, 0)$, $p1 = (0, 0, 0)$, $r2 = (1, 0.8, 0)$, $p2 = (-0.3, 0, 0)$.

Conclusion

From the analysis of the collected data we can conclude that the comparisons show that the UCCSD variational form is better than EfficientSU2, which is much more difficult to optimize. Also, the COBYLA Optimizer is more efficient than SLSQP, but the SPSA optimization method is what best helps in convergence of results. The curve given by the noise-mitigation technique follows very well the data obtained from the run on real QC, reproducing it almost perfectly for both the processors. Lastly, the simulations show a feasibility of combination of method of quasi-classical trajectories with VQE.

Future outlook

This project was inspired by a very hot-topic in nowadays chemistry - molecural dynamics performed on hybrid quantum-classical architectures. Thus, we hope to improve on it by further analysis of the problems and their substantial components like a choice of efficient optimizers and variational forms. We also hope to continue in that direction, making more robust conclusions from our computations and possibly focusing also on optimization of quantum circuit properties (e.g. their depth). Last but not least, we aim to simulate the dynamical system with additional (mitigated) noise as the very next step.

References

[1] Qiskit SPSA Optimizer. qiskit.aqua.components.optimizers.SPSA - Qiskit 0.29.0 documentation. (n.d.).

[2] Peruzzo, A., McClean, J., Shadbolt, P., Yung, M. H., Zhou, X. Q., Love, P. J., ... & O'brien, J. L. (2014). A variational eigenvalue solver on a photonic quantum processor. Nature communications, 5(1), 1-7.

[3] McClean, J. R., Romero, J., Babbush, R., & Aspuru-Guzik, A. (2016). The theory of variational hybrid quantum-classical algorithms. New Journal of Physics, 18(2), 023023.

[4] Porter, R. N., Raff, L. M., & Miller, W. H. (1975). Quasi-classical selection of initial coordinates and momenta for a rotating Morse oscillator. The Journal of Chemical Physics, 63(5), 2214-2218.

[5] Nagy, T., Vikár, A., & Lendvay, G. (2018). A general formulation of the quasiclassical trajectory method for reduced-dimensionality reaction dynamics calculations. Physical Chemistry Chemical Physics, 20(19), 13224-13240.

[6] Fedorov, D. A., Otten, M. J., Gray, S. K., & Alexeev, Y. (2021). Ab initio molecular dynamics on quantum computers. The Journal of Chemical Physics, 154(16), 164103.

PRACE SoHPCProject Title

Molecular Dynamics on Quantum Computers

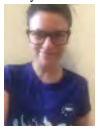
PRACE SoHPCSite

IT4Innovations National Supercomputing Centre, Czech Republic

PRACE SoHPCAuthors

Jenay Patel, University of Nottingham, UK
Carola Ciaramelletti, Università degli studi dell'Aquila, Italy

Jenay Patel



Carola Ciaramelletti

PRACE SoHPCMentors

Martin Beseda, IT4Innovations National Supercomputing Centre, Czech Republic
Stanislav Paláček, IT4Innovations National Supercomputing Centre, Czech Republic

PRACE SoHPCContact

Jenay Patel, University of Nottingham
E-mail: jenaypatel@live.com
Carola Ciaramelletti, Università degli studi dell'Aquila
E-mail: ciaramelletticarola@gmail.com

PRACE SoHPCSoftware applied

IBM's Qiskit

PRACE SoHPCMore Information

www.qiskit.org
www.quantum-computing.ibm.com

PRACE SoHPCAcknowledgement

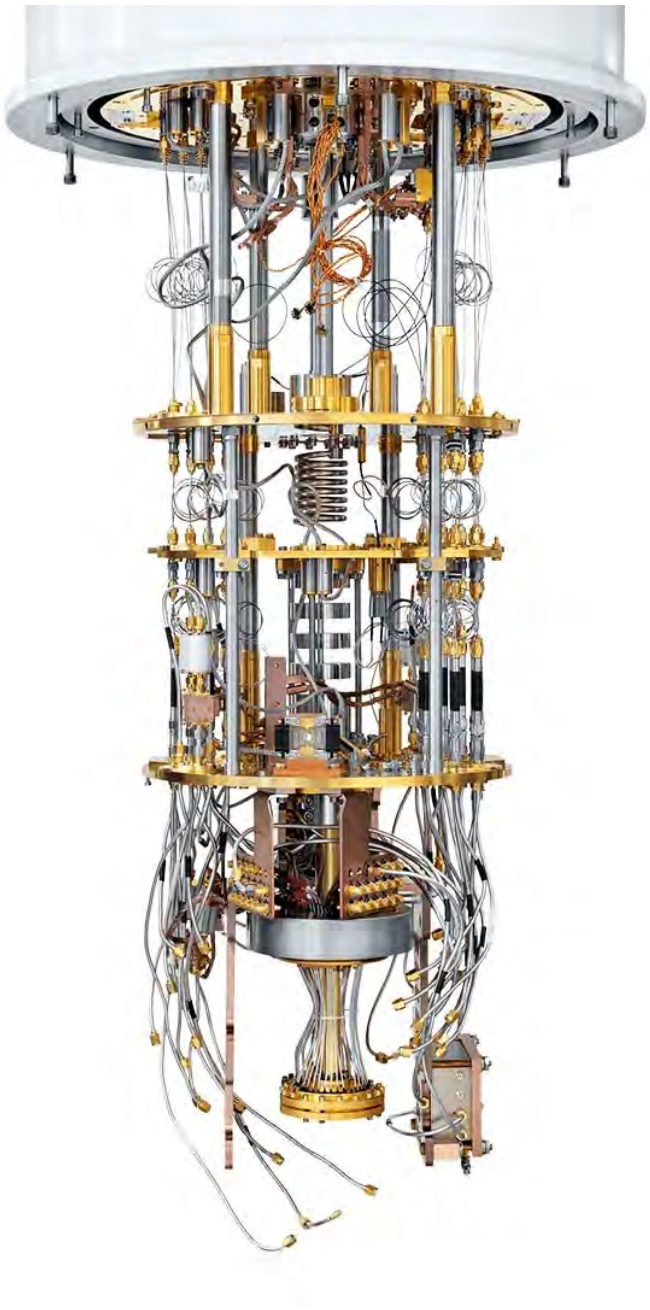
We are very grateful to our mentors, Martin and Stanislav for their support and attention: thank you for choosing us. Special thanks go to Assoc.Prof.Dr. Leon Kos for being a point of reference. Thanks also to Karina Pešatová for helping with the blog!

PRACE SoHPCProject ID

2118

Breaking the Bounds using quantum algorithms

Lucía Absalom Bautista
Spyridon-Andreas Siskos



In this project we have focused on studying quantum computer theory and quantum algorithms such as Grover, Deutsch-Jozsa and Shor, implementing the last one to factorize numbers, which can be used to break the most famous encryption algorithms, RSA.

Quantum computing is a well-established research area on the international scientific scene.

Anyone who has heard of it knows about its potential and the revolution that it will bring in the coming years. In our project we start from scratch by defining the concept of qubit, the quantum version of the classic binary bit physically realized with a two-state device. If you have one qubit and you ask yourself: "What can I do with it?". The answer is clear, you can only change its state or you can measure it. But what if we had two qubits? Then we could do all of the above as well as create qubits with the special feature of being entangled. This means that if I measure one qubit, I don't need to measure the other, as its result is the same or opposite (it depends on the type of entanglement used). This amazing phenomenon is harnessed by quantum teleportation, which is the basis for quantum cryptography.

Quantum computing basics

Qubits

The two basics states of qubits are $|0\rangle$ and $|1\rangle$ corresponding to the 0 and 1 of the classic bit. But in addition, qubits can be in a state of quantum superposition combining these two states $(\alpha|0\rangle + \beta|1\rangle)$. An intuitive graphical representation of a qubit can be achieved using the Bloch sphere, shown in Fig. 1. Apart from the states $|0\rangle, |1\rangle$, there are other possible general states of type $|\Psi\rangle$.

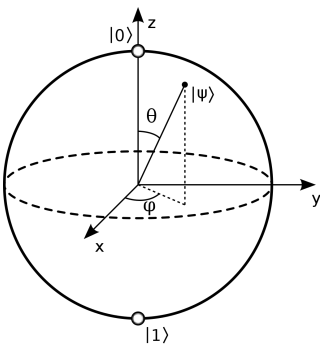


Figure 1: The Bloch sphere

Gates

Quantum computing uses different properties of quantum states to perform computation:

- **Superposition:** is one of the fundamental principles of quantum mechanics and it can be seen as a linear combination of all quantum states.

- **Entanglement**: A pair of qubits is entangled when the quantum state of each particle cannot be described independently of the quantum state of the other particle. In other words, the outcome of the measurement of one qubit will always affect the measurement of the other qubit.

How do we achieve these states? Using quantum gates whose operations with one or more qubits can be described by linear algebra. As quantum theory is unitary, quantum gates are represented by unitary matrices.

Using the concept of the Bloch Sphere, we can visualise the gates as the composition of rotations around the X, Y and Z axes. The most commonly used gates are Hadamard gate, Paulis X, Y and Z gates and CNOT gate. Using linear algebra, the matrix describe for an X gate would be: $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.

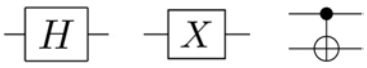


Figure 2: Representation of Hadamard, X, and CNOT gate

Quantum circuits

Quantum circuits are an ordered sequence of quantum gates and measurements, all of which may be conditioned on and use data from the real-time classical computation. They can be represented graphically as shown for example in Fig. 3.

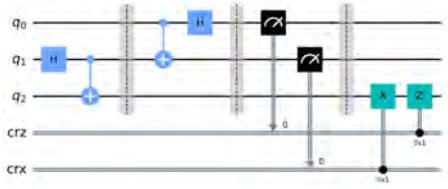


Figure 3: The quantum teleportation circuit

Deutsch-Jozsa algorithm

It was the first example of a quantum algorithm that performs better than the best classical algorithm.

Problem: We are given a unknown Boolean function f i.e. given a string of bits it returns either 0 or 1:

$$f(\{x_0, x_1, ...\}) \rightarrow 0 \text{ or } 1$$

where x_n is 0 or 1 and this function is either balanced or constant, meaning that a constant function will return 0's or 1's for all possible inputs and a balanced function will have 0's for half of the inputs and 1's for the other half. Is our function f balanced or not?

Solution: In a classical computer in the best case scenario we would need at least 2 queries to the oracle to determine whether the function is balanced or not. In the worst case we need half the number of inputs plus one. The power of quantum computation lies in the fact that with a single query to the oracle we are able to know if our function is balanced or not. Our function $f(x)$ implemented in the oracle is described as follows

$$f : |x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$$

where $\oplus$ is addition modulo 2 (XOR).

So let's mathematically derive how the algorithm circuit (Fig.4) works!

1. We get two quantum registers, one with n qubits initialized to $|0\rangle$ and a single qubit initialized to $|1\rangle$.

$$|\psi_0\rangle = |0\rangle^{\oplus n}|1\rangle$$

2. We apply a Hadamard gate to each state to create superposition state.

$$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle(|0\rangle - |1\rangle)$$

3. We apply the quantum oracle.

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)$$

4. We then apply a Hadamard gate to each qubit in the first register.

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \left[\sum_{y=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle \right]$$

$$= \frac{1}{2^n} \sum_{x=0}^{2^n-1} \left[\sum_{y=0}^{2^n-1} (-1)^{f(x)} (-1)^{x \cdot y} \right] |y\rangle$$

where $x \cdot y = x_0 y_0 \oplus x_1 y_1 \oplus ... \oplus x_{n-1} y_{n-1}$

5. From the last formula it can be seen that the probability of measuring $|0\rangle^{\oplus n}$ is

$$\left| \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \right|^2$$

so if f is constant, then this probability is 1. But if f is balanced, then the probability of measuring $|0\rangle^{\oplus n}$ is 0.

Hence, if we measure all zeros, we can say with certainty that f is constant.

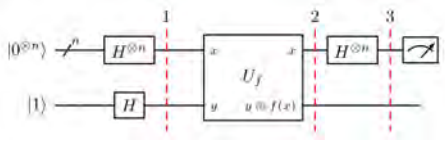


Figure 4: Deutsch-Jozsa algorithm circuit

Grover's algorithm

Grover's searching algorithm shows us the superior speed searching of quantum algorithms in database which can be speeded up quadratically comparing it to the classical algorithm.

Problem: We are given a list of N items and we want to search for one special item, let's call it a. Our task is to find the item a.

Solution: To find item in an unstructured database with a classical algorithm, we would have to check on average N/2 times and in the worst case scenario we would have to check N times. With quantum computers and Grover's algorithms we can find the solution with only $\sqrt{N}$ steps.

Grover's algorithm solves Oracles that add a negative phase to the solution states:

$$U_w|x\rangle = \begin{cases} |x\rangle & \text{if } x \neq a \\ -|x\rangle & \text{if } x = a \end{cases}$$

So a way of implementing this Oracle would be

$$U_w|x\rangle = (-1)^{f(x)}|x\rangle$$

If we want to find the item a in our database, we have to construct the oracle in such a way to give us as a return value the item that we want to find. To be more specific, when the state of the item a comes into the oracle, we want to flip its amplitude value in the negative phase to the solution states. With this method, we are going to have some solutions but not only the optimum.
In order to get the optimum solution, we have to apply the Grover's diffusion operator that it will help us determine the final solution.

Shor's algorithm and number factorization: breaking RSA.

Introduction

Shor's Algorithm is one of the best known quantum algorithms, because it can be used for fast number factorization. It's an hybrid algorithm meaning that part of it can be executed in a classical computer and another one in a quantum computer. In this way, we take advantage of both classical and quantum computers.

So why is Shor's algorithm so powerful?

The best algorithm for number factorization in classical computers belongs in NP class, but Shor's Algorithm $\mathcal{O}(b^3)$ in complexity time and $\mathcal{O}(b)$ in space complexity, which means that it can solve our problem in polynomial time.

One of its well-known applications is in cryptography and how it can be used to break the RSA encryption, that it is most common cryptographic algorithm that it is used all over the world.

In this section we are going to study number factorization for numbers, 35 and 91. The solutions provided for number 35 was run in a real quantum computer, but the one provided for number 91 was run in a simulator because it requires using a number of qubits that IBM free quantum computer doesn't support.

Explanation

Problem: Given an integer number N, we want to find another integer p between 1 and N that divides N.

Solution:

1. Choose an integer number a so, that $1 < a < N$.
2. Compute $\gcd(a, N)$ (this can be done using Euclid's algorithm).
3. If $\gcd(a, N) \neq 1$, then a is a non trivial factor of N. **Finish.**
4. If not use the period-finder quantum program to find r, the period of the following function.

   $$f(x) = a^x \bmod N$$

   i.e. the smallest integer r for which

   $$f(x + r) = f(x)$$

5. If r is odd go to step 1
6. If $a^{(r/2)} \equiv -1 \bmod N$ go to step 1
7. The factors of N are

   $$\gcd(a^{(r/2)} \pm 1, N)$$

   Finish.

Let's start with number 35. For that we choose $a = 6$. We use the quantum algorithm to compute the period r.

Shor's algorithm for factorization of number 35

The circuit described below finds the period, r, that is needed in **step 4** of the general factorization algorithm. For this we use 5 qubits.

1. The first step is to apply Hadamard gate (H), to the first 2 qubits, in order to have them in superposition and a XOR gate to the third one in order to use it in the two custom functions of modulo that we built.
2. Then we used the 2 custom functions(oracles) that we built for $6^1 \bmod 35$ and $6^2 \bmod 35$, taking as an input the first and the second qubit which are in superposition for each case.
3. The last step of the algorithm is to run an Inverse Quantum Fourier Transform (QFT†) on the 2 first qubits and then measure them.

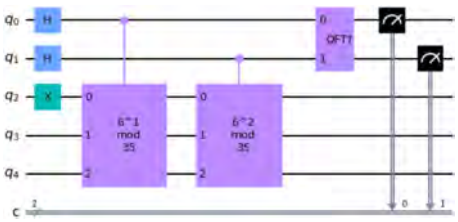


Figure 5: Period-finder circuit for $N = 35$, $a = 6$.

The Oracle described in step 2 works with three qubits and in a first step it changes qubits q_0 and q_2, then we apply an X gate in qubit q_1.

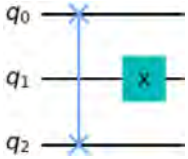


Figure 6: $6^1 \bmod 35$ Oracle

Measured Results

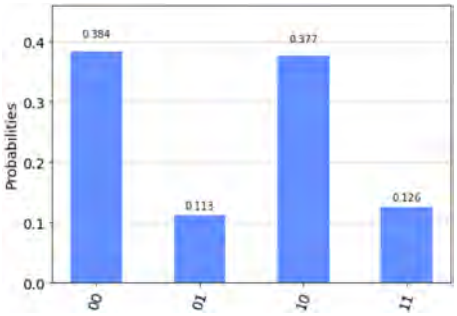


Figure 7: Results obtained by running on IBM quantum computer *imbq_quito*.

The value whose probability stands out apart from 0 is 10(2). We can take this as the r period because the search period r is calculated as the ratio of the total number of possible outcomes, which in this case is 4, and the smallest non-zero outcome, which in this case is 2.

Also, this period r can be determined as the number of peaks of the probability histogram, which, as can be seen, is again the number 2. Outputs 01 and 11 with smaller probabilities are due to the inaccuracy of the quantum computer used, so we neglect them. If we used a simulator their probability would be zero.

Ending the algorithm

For that we follow the steps 5, 6 and 7:

Step 5

$r = 2$, so we continue to step 6.

Step 6

$6^{2/2} \equiv -1 \bmod 35$

The equation above doesn't verify so we continue to step 7.

Step 7

We calculate the factors:

- $\gcd(6 + 1, 35) = \gcd(7, 35) = 7$
- $\gcd(6 - 1, 35) = \gcd(5, 25) = 5$

Our solutions are numbers 7 and 5 and we can see that $7 \times 5 = 35$ so ...

We have succeeded!

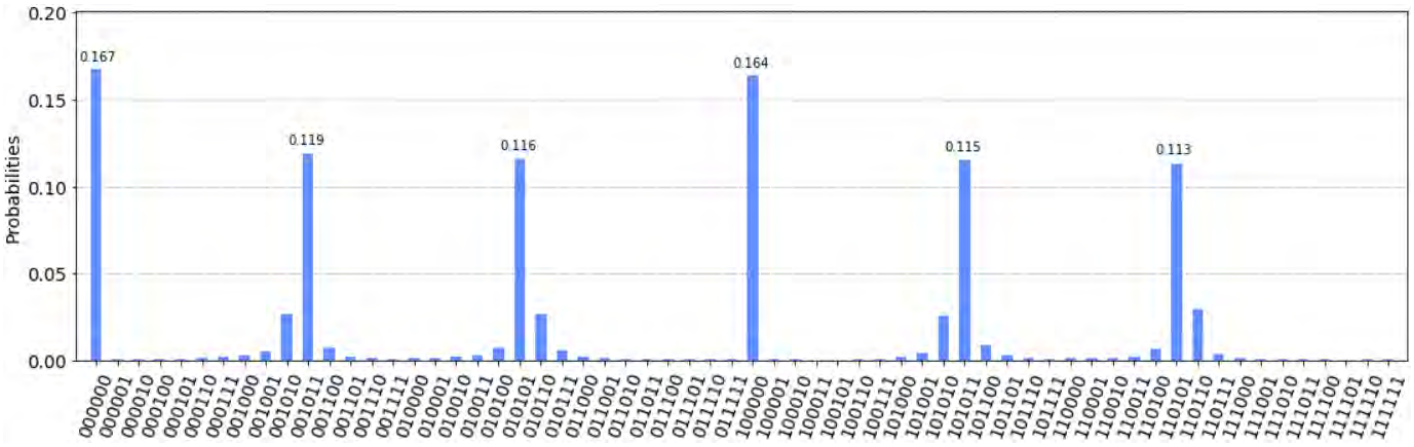


Figure 8: Results obtained by running on IBM quantum simulator *imbq_qasm_simulator*.

Shor's algorithm for factorization of number 91

This algorithm only differs from the previous one in the number of qubits and the oracles. In figures 10 and 9 we can see how we implemented the circuit and the oracle.

The value of the variable a should be chosen as small as possible, so $a = 4$ was the best choice. Fig. 9 shows a circuit simulating the $4^x \bmod 91$ function.

When used repeatedly, this circuit generates successively the values 4, 16, 64, 74, 23, 1, 4, 16, 64, 74, 23, 1, 4, 16, ... etc., which corresponds exactly to the function of $4^x \bmod 91$. The truth table of this circuit is shown in 1.

Table 1: $4^x \bmod 91$ function truth table.

Input	Output
0000001 (1)	0000100 (4)
0000100 (4)	0010000 (16)
0010000 (16)	1000000 (64)
1000000 (64)	1001010 (74)
1001010 (74)	0010111 (23)
0010111 (23)	0000001 (1)

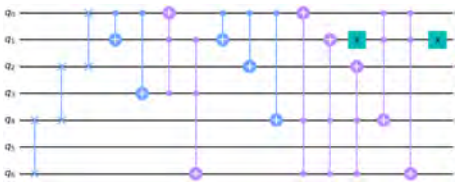


Figure 9: $4^1 \bmod 91$ Oracle.

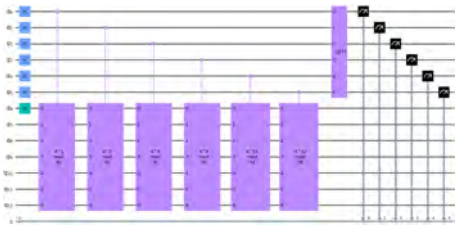


Figure 10: Period-finder for $N = 91$, $a = 4$.

Measured results

From histogram in Fig. 8 we can see clearly that the highest probability value have 6 peaks (0, 11, 21, 32, 43, and 53), so the period $r = 6$.

The period r can also be calculated as the ratio of the total number of possible outcomes, which in this case is 64, and the smallest non-zero outcome, which in this case is 11. However, in some cases (like this one) an integer number may not come out. Then it is sufficient to round the result and make sure that this rounded value matches the number of histogram peaks.

Ending the algorithm

For that we follow the steps 5, 6 and 7:

Step 5

Looking at the peaks of the histogram we see $r = 6$.

Step 6

$4^{4/2} \equiv 64 \equiv -1 \bmod 91$

The equation above doesn't verify so we continue to step 7.

Step 7

We calculate the factors:

- $\gcd(4^3+1, 91) = \gcd(65, 91) = 13$
- $\gcd(4^3-1, 91) = \gcd(63, 91) = 7$

Our solutions are numbers 13 and 7 and we can see that $13 \times 7 = 91$ so...

We have succeeded!

Conclusions

In this summer internship we have experienced building quantum circuits of concrete examples, running them on simulators, as well as on real quantum computers!

We have verified that the real practical use of quantum computers is currently hampered mainly by their inaccuracy and the small number of qubits. If this obstacle is removed in the future, the quantum computers will play a key role in high-performance computing.

References

[1] Nielsen, M. A. and Chuang, I. L. (2011). Quantum Computation and Quantum Information.

[2] Coles, J. P. and Eidenbenz, S. ... +29 authors (2018). Quantum Algorithm Implementations for Beginners.

PRACE SoHPCProject Title

Quantum algorithms and their applications

PRACE SoHPCSite

IT4Innovations, VŠB - Technical University of Ostrava, Czech Republic

PRACE SoHPCAuthors

Lucía Absalom Bautista, University of Seville, Spain

Spyridon-Andreas Siskos, University of Crete, Greece

Lucía Absalom Bautista

Spyridon-Andreas Siskos

PRACE SoHPCMentor

Jiří Tomčala, IT4Innovations, VŠB - Technical University of Ostrava, Czech Republic

PRACE SoHPCContact

Jiří Tomčala, IT4Innovations, VŠB - Technical University of Ostrava
E-mail: jiri.tomcala@vsb.cz

PRACE SoHPCSoftware applied

Qiskit

PRACE SoHPCMore Information

quantum-computing.ibm.com

PRACE SoHPCAcknowledgement

We acknowledge the use of IBM Quantum services for this work.

PRACE SoHPCProject ID

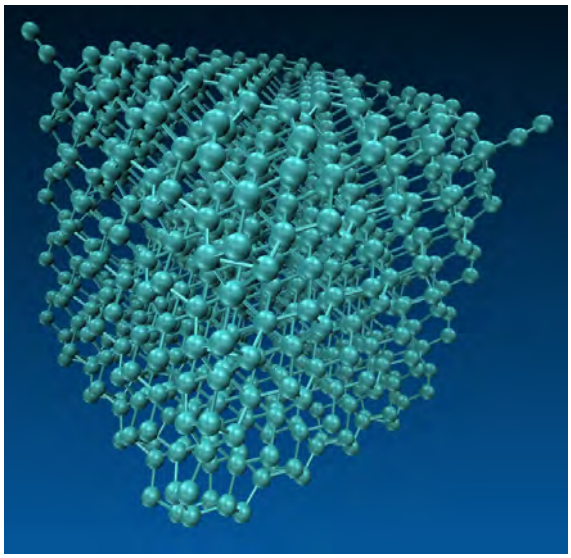
2119

Implementation of a Carbon nanostructures simulation library for the Hubbard model.

Building a CNS software

Marc Túnica

This project aims to design and develop a software to model carbon nanostructures. Particularly, a simulation of the Hubbard Model applied to carbon nanostructures.



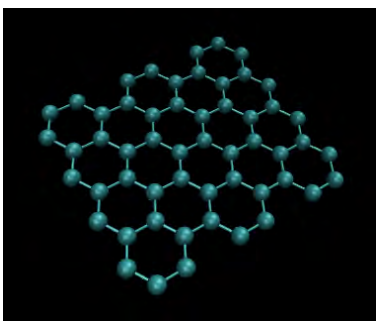
Diamond Carbon structure.

CNS simulation in a nutshell

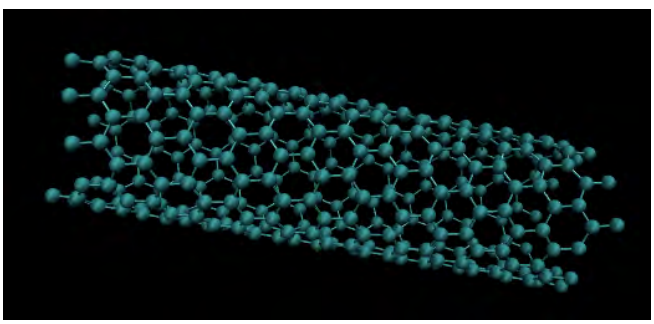
The **carbon nanostructures (CNS) simulation** is the simulation and modelling of carbon nanostructured systems, their properties and behaviour in particular conditions. Roughly speaking, we can define a carbon nanostructure as a structure, composed exclusively by carbon atoms, in which at least one of the three spatial dimensions is of the order of hundred or ten nanometers (remember that $1nm = 10^{-9}m$). There are several types of carbon nanostructures, with different shapes, organisation of the atoms and, hence, an enormous variety of properties.[2] The most familiar is the **graphene** (figure 1a). The graphene is a bi-dimensional layer (a layer with the thickness of an atom) arranged in a honeycomb lattice. It is a material characterised by its efficient thermal and electrical conductivity. Also is flexible and hard, so has an infinite number of possible applications. We shall mention **carbon nanotubes** (CNT) (figure 1b). This particular nanostructure consists of a carbon atoms honeycomb lattice which is enrolled forming a tube with a small diameter (about ten or hundred nanometers). Another important carbon nanostructure is the **fullerene**, that can be described as the typical soccer ball (figure 1c).

The connection between **High Performance Computing** (HPC) and simulation CNS becomes apparent considering some details about the way such a simulation is implemented. Key operations come from the branch of linear algebra (matrix-matrix, matrix-vector operations, inversion of matrices, determinants, and many more) in large dimensions (greater than 100) which can utilise the continuously increasing advances in HPC related fields. This project aims to implement a CNS simulation library with **CPU**, **GPU**, **multi-CPU** and **multi-GPU** support. Specifically, we want to base our framework in the **Hubbard model**, that is a simple theoretical model which describes the interaction between particles in a lattice. The main objective is to design a software with some linear algebraic operations that can operate with fermion matrices and time vectors. A fermion matrix is an operator which contains the information of a fermion system, that is a physical system formed by **fermions** (common matter particles). According to the Hubbard model, we can describe a CNS system using this sort of matrices. Since this is an educational and informative report, we will not provide an extensive explanation about fermion matrices and the Hubbard Model, however, the reader can find some useful information in numerous references.[3,4] In our model, we also apply the discretization of space-time. The time, for our purpose, is not continuous. It is divided in N_t parts. The distance between each time-step is δt (small enough to be a good approximation). Then, our time encompasses $t = \{0, 0+\delta t..., N_t - \delta t, N_t\}$. Dropping the dimensions as all computations are done without units, we store the information in a N_t-dimensional time-vector. Hence, $t \rightarrow t = \{0, 1, ...N_t - 1, N_t\}$. The spatial directions are naturally discretized by the ion grid, with each point position corresponds to the position of a particle. Then, the **spatial volume** corresponds to the number of carbon atoms of our system.

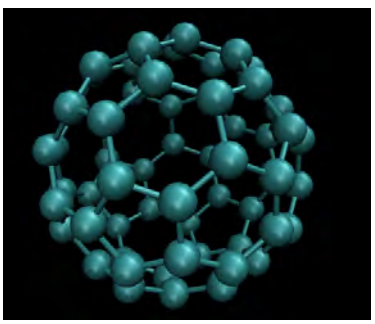
Let's introduce a few notation based on



(a) Graphene



(b) Nanotube



(c) Fullerene

Figure 1: Most familiar nanostructures images. Plotted by the VMD software.[1]

Block	Function	Input	Output
Tensor class[1]	Operator *	Tensor and a number	Tensor multiplied by the number.
		Two tensors	Element wise product of the two tensors.
	Operator[]	A succession of integer numbers (indexes).	The sub-tensor or element of the index given.
	Shape	A integer number (i).	The i-dimension size.
	Exponential	A tensor.	The exponential of the tensor.
	Expand	A tensor and a dimension (dim).	A tensor with an extra dimension of value dim.
	Shift	A tensor, an integer (positions) and a number (boundary conditions).	The tensor shifting a number (positions) of positions and multiplying periodic elements by a boundary condition.
Linear Algebra[1]	Mat.-vec. product	Two Tensors.	A tensor equals to the product of the inputs.
Fermion Matrix	BF	A tensor ($e^{(\tilde{\kappa}-\tilde{\mu})}$) and a Time-Tensor ($\phi$).	A Time-tensor equals $\mathcal{B}_{t'}e^{i\phi} \times e^{(\tilde{\kappa}-\tilde{\mu})}$.
	Exp.-disc.	A tensor ($e^{(\tilde{\kappa}-\tilde{\mu})}$) and two Time-Tensors ($\phi, \psi$).	A Time-Tensor solution of (4).

Table 1: Table some of the relevant functions implemented in the library. **Mark 1:** Symmetric function for Time-Tensor.

the references *Wynen et al.*[3] and the *isle documentation*.[4] The **fermionic action** is an abstract quantity which describes the overall physical system of the fermions.[5] It can be described by the formula

$$S = -\log\det M\left(\phi,\tilde{\kappa},\tilde{\mu}\right) \cdot M^{T}\left(-\phi,\sigma_{\tilde{\kappa}}\tilde{\kappa},-\tilde{\mu}\right), \quad (1)$$

where $\tilde{\kappa}$ is the **hopping matrix**,[6] $\tilde{\mu}$ is the **chemical potential** and $\sigma_{\tilde{\kappa}} = \pm 1$. M is a matrix (fermion matrix) and M^T denotes the transposed matrix. M, in our implementation,

$$M\left(\phi,\tilde{\kappa},\tilde{\mu}\right)_{x't',xt} = \delta_{x',x}\delta_{t',t} - \mathcal{B}_{t'}F_{t'}\left[\phi,\tilde{\kappa},\tilde{\mu}\right]_{x',x}\delta_{t',(t+1)}, \quad (2)$$

where $\mathcal{B}_{t'} = 1$ if $t \neq 0$ and -1 if $t = 0$. This variable lets us to encode the anti-periodic boundary condition. $\delta_{x',x}$ and $\delta_{t,t'+1}$ are Dirac's deltas. $F_{t'}\left[\phi,\tilde{\kappa},\tilde{\mu}\right]_{x',x} = \left(e^{\tilde{\kappa}-\tilde{\mu}}\right)_{x',x} e^{i\phi_{xt}}$. Here, $\left[e^{\tilde{\kappa}-\tilde{\mu}}\right]_{x',x}$ and $e^{i\phi_{xt}}$ are a $N_x \times N_x$ and a $N_x \times N_t$ matrix respectively. We suppose that N_x is the spatial volume (number of carbon atoms) and N_t the temporal dimension. Summing up, the problem is reduced to calculating the expression

$$G_{x't',xt} = \left[e^{\tilde{\kappa}-\tilde{\mu}}\right]_{x',x} e^{i\phi_{xt}}\delta_{t',(t+1)}. \quad (3)$$

Imagine another situation. We want to multiply expression (2) by a tensor ψ^{xt} of dimension $N_x \times N_t$. Note that the word **tensor** does not refer the mathematical object but simply to a multidimensional matrix. Then,

$$M\psi^{xt} = \sum_{x=0,t=0}^{N_x,N_t} \delta_{x',x}\delta_{t',t}\psi^{xt} - \mathcal{B}_{t'}G_{t'x',tx}\psi^{xt}. \quad (4)$$

Thus, because of the Dirac's delta $\delta_{t',(t+1)}$, the most complex expression to evaluate is

$$\sum_{t=1}^{N_t} G_{x't',xt}\psi^{xt} = G_{x't',xt}\psi^{xt}, \quad (5)$$

and this is one of our principal goals.

Structure of the library

But how is our library organised? In **C++**, a **class** is a block which holds its own data variables and functions used to manipulate these variables, which can be accessed and used by creating an instance of that class.[7] Our software, written in C++ based on the **torch** library,[8] implements two classes (so two distinct kind of tensors). The first one, the **Tensor class**, is only a spatial tensor (without time variable). The other one is the **Time-Tensor class**. In this latter case, an extra dimension (discretization of time) is stored. Both classes have the same functions and a variable (from now namely data) that is a $torch :: tensor$. All the information is stored in data. In addition, we implement also more functions distributed in two namespaces, termed **Linear Algebra** and **Fermion Matrix**. A **namespace** is a declarative region that provides a scope to the identifiers (function, variables, ...) inside it.[9] The "Linear Algebra" namespace incorporates some functions related to matrix linear algebra operations. On figure 1, most relevant functions are explained. Let's talk about some definitions that are interesting to understand the algorithms.

- Given a tensor, we **multiply it by a scalar** if we multiply each element of the tensor by the scalar.
- The **element wise product of two tensors** is the multiplication of each element of the tensor by the element in the same position of the other one (of course, it is mandatory that both tensors have the same dimension).
- The **element wise exponential** of a tensor is the exponential of each element of the tensor.

Within all the functions implemented by the two classes, the most interesting is the called **shift function**. The shift function allows the translation of all the element of a tensor. For example, let ψ be a n-dimensional vector $\psi = (\psi_1, ...\psi_n)$, if we shift one position, we obtain $\psi = (\psi_n, \psi_1, ..., \psi_{n-1})$. This function is motivated by the computation of expression (5). Computationally, We can obtain $\psi^{x,(t+1)}$ applying the shift function.

Regarding "Fermion Matrix" namespace, "**BF**" function is the responsible to calculate expression (3). The input must be two matrices ($\left[e^{\tilde{\kappa}-\tilde{\mu}}\right]_{x',x}$ and ϕ_{xt}). It reproduces, for every time-step, the following algorithm:

1. Calculate $i \cdot \phi_x$. Remember that for a particular fixed time, ϕ_x is a N_x-dimensional tensor.
2. Calculate the element-wise exponential of the preceding result.
3. Expand the solution by the **expansion** function ("expand"), so we obtain a $N_x \times N_x$ tensor.
4. Then, we are able to apply the element wise product by $\left[e^{\tilde{\kappa}-\tilde{\mu}}\right]_{x',x}$, since its dimension is also $N_x \times N_x$.
5. We repeat the process for each time. At the end of the loop, a $N_t \times N_x \times N_x$ tensor is returned.
6. To apply boundary conditions ($\mathcal{B}$), we multiply by -1 the first time-row (the $N_x \times N_x$ tensor corresponding to $t = 0$).

We do not stop here. To compute equation (4), we must apply the subsequent algorithm, corresponding to the function **Exponential Discretisation** ("exp.-disc.") from the "Fermion Matrix" namespace:

1. We determine the solution of equation (3) with the previous algorithm. Let's denote the solution by M, which is a $N_t \times N_x \times N_x$ Tensor. The next steps, have to be applied for each time.

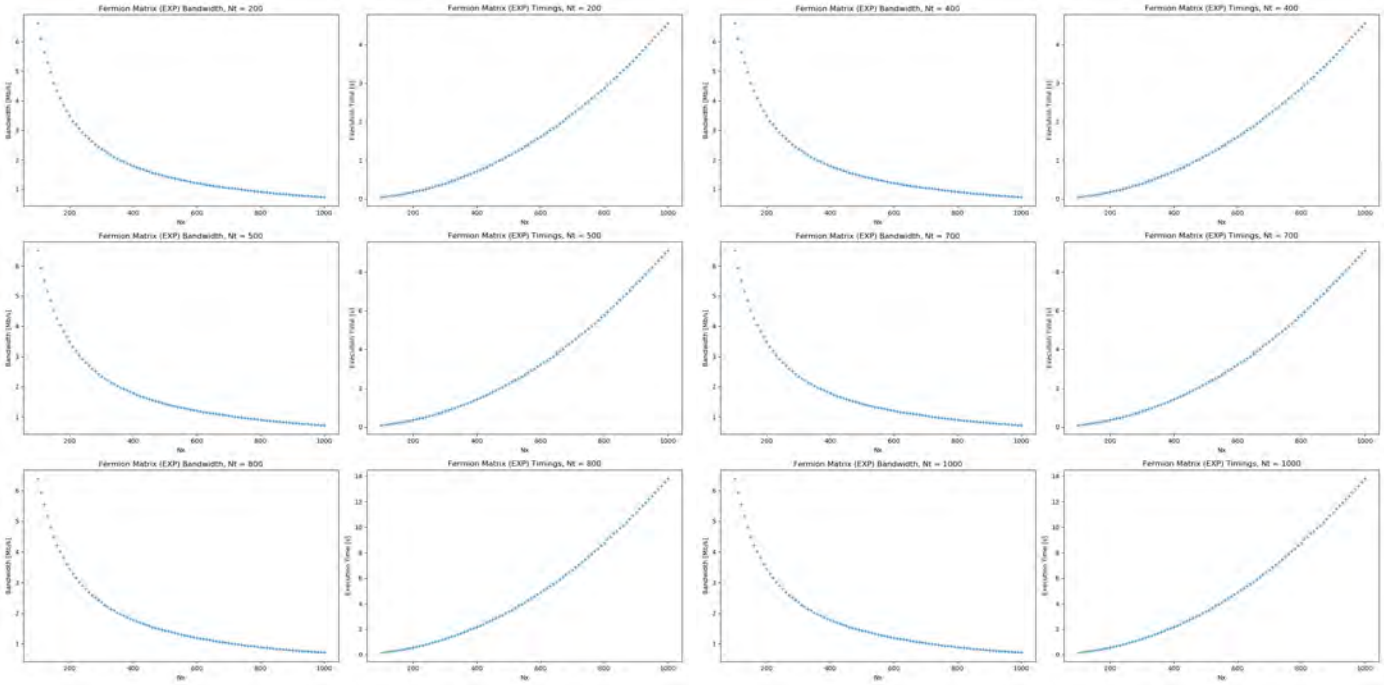


Figure 2: Benchmark results. For each temporal dimension $N_t = \{200, 400, 500, 700, 800, 1000\}$, plot of the bandwidth(in mb/s) as a function of the number of carbon atoms(N_x) and the time execution (in seconds) as a function of the number of carbon atoms.

2. We shift one position tensor ψ^x. So, for a particular time, we take a N_x tensor.
3. We multiply this shifted tensor by a $N_x \times N_x$ tensor, M, at a particular time. We use **matrix vector multiplication** function ("mat-vec").
4. We repeat steps 2 and 3 for each time. At the end, a $N_t \times N_x$ tensor is obtained. With our notation, $G_{x't',xt} \cdot \psi^{xt}$.
5. Return the difference between $N_t \times N_x$ tensors, $\psi^{xt} - G_{x't',xt}\psi^{xt}$.

Benchmark and conclusions

The proper and easiest way to benchmark the program is through some examples. Nonetheless, we created random tensors of different dimensions in order to obtain the fermion matrix and multiply it by a Time-Tensor (ExpDisc function). It provided us a general vision of how the program is working, even, without examine a particular case. We used the benchmark library **google/benchmark**.[10] The aim of the benchmark is to compare our implementation with other ones and select the better optimised. For our torch implementation, the outcomes are showed on figure 2. On the right column of each cell (there are two cells at each row), the time of execution per the number of carbon atoms per each temporal dimension is plotted. The reader can observe that the time increases with the number of atoms and executions are slower for sizeable temporal dimensions. Evidently, this tendency is expected, so at larger volumes and greater temporal dimensions, more operations are executed, raising the time of execution. On the left column, the bandwidth as a function of the number of atoms is represented (also for each temporal dimension). The bandwidth is the rate of data transmitted over a path. It is the volume of information transferred and decreases when space volume increases. The main hypothesis is that large-scale tensors spend more memory, and the bandwidth is reduced. The reduction of the bandwidth is extremely fast and can be a problem for real-live modelling, because we need to model systems with a huge amount of particles. Although, benchmark conclusions are suitable with other simulations programs such as **Isle**.[4] We can not determine if torch is the proper library, since we do not have relevant data of counterpart implementations yet. Nevertheless, it can not be rejected. To sump up, this report operates as a preliminary first benchmark and shows that torch implementation is a feasible possibility to simulate carbon nanostructures.

References

[1] VMD: Visual Molecular Dynamics. https://www.ks.uiuc.edu/Research/vmd/. Date Accessed 01-08-2021.

[2] O. A. Shenderova, V. V. Zhirnov, and D. W. Brenner. Carbon Nanostructures. *Critical Reviews in Solid State and Materials Sciences*, 27(3-4):227–356, 2006.

[3] Jan-Lukas et al. Wynen. Avoiding Ergodicity Problems in Lattice Discretizations of the Hubbard Model. *Physical Review B*, 100(7), dec 2018.

[4] Evan Berkowitz et al. isle/docs at devel · evanberkowitz/isle. https://github.com/evanberkowitz/isle/tree/devel/docs. Date Accessed 30-08-2021.

[5] Action | physics | Britannica. https://www.britannica.com/science/action-physics. Date Accessed 04-08-2021.

[6] Maher Ahmed. Understanding of hopping matrix for 2D materials taking 2D honeycomb and square lattices as study cases. oct 2011.

[7] C++ Classes and Objects - GeeksforGeeks. https://www.geeksforgeeks.org/c-classes-and-objects/. Date Accessed 01-09-2021.

[8] PyTorch. https://pytorch.org/. Date Accessed 01-08-2021.

[9] Namespace in C++ | Set 1 (Introduction) - GeeksforGeeks. https://www.geeksforgeeks.org/namespace-in-c/. Date Accessed 01-09-2021.

[10] google/benchmark: A microbenchmark support library. https://github.com/google/benchmark. Date Accessed 03-08-2021.

PRACE SoHPC

High Performance Quantum Fields

PRACE SoHPCSite

Jülich Supercomputer Center, Germany

PRACE SoHPCAuthor

Marc Túnica, University of Barcelona, Spain

Marc Túnica

PRACE SoHPCMentor

Marcel Rodekamp, JSC, Germany

PRACE SoHPCContact

Marc, Túnica, University of Barcelona
E-mail: infomtunica@gmail.com

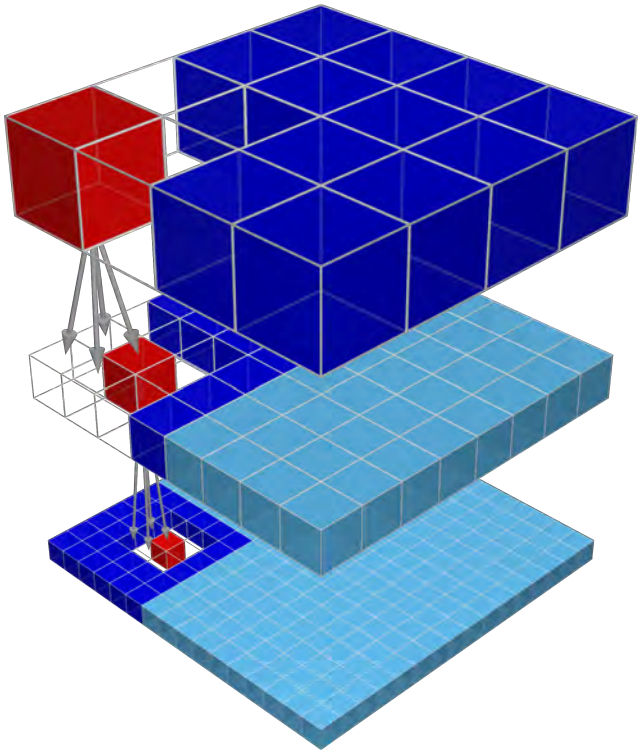
PRACE SoHPCAcknowledgement

I would like to extend my most sincere gratitude to Marcel Rodekamp, for his patience and his guidance during the SoHPC. I also would like to thank Dr. Evan Berkowitz, Dr. Eric Gregory and Dr. Stefan Krieg their help and advice.

PRACE SoHPCProject ID

2120

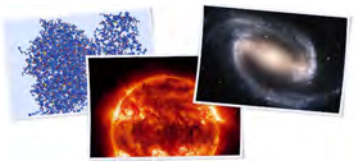
Tiny, tiny, tasks! Huge Impact?



Arthur Guillec & Tristan Michel

Current hardware theoretically allows us to perform an incredible amount of floating point operations per second (FLOPS). However, it is very difficult to fully utilise this potential. We will therefore measure how much of the resources of a computer a given tasking framework manage to use in the context of the Fast Multipole Method (FMM).

First of all, what is the Fast Multipole Method (FMM)? It is a technique developed to accelerate the calculation of long-range forces in the N-body problem. There are many applications in molecular dynamics, plasma physics and astrophysics.



Naive solving methods are extremely limited in application because of their $O(n^2)$ complexity. The FMM and its complexity in $O(n)$ is therefore a must in solving the N-body problem when there are many interacting entities, it is even been said to be one of the top ten algorithms of the 20th century.

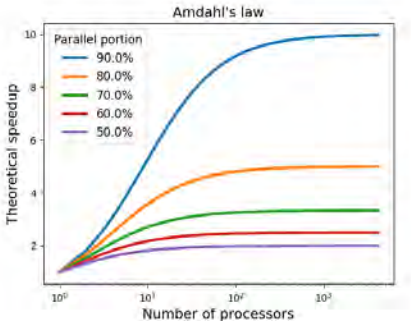
However, for larger inputs, FMM is still very time consuming. Since this method is used at every time step of a numerical solving algorithm, the execution time must be extremely low. So there is only one solution to reduce the execution time: parallelizing the program.

JÜLICH FORSCHUNGSZENTRUM

To achieve this goal, we started working with a sequential implementation of the FMM written by a team at the Jülich Research Centre. The program is divided into 5 steps that we modified to accommodate a tasking framework called `HPX`, a general purpose `C++` runtime system for parallel and distributed applications developed by the STE||AR group at Louisiana State University.

HPX

Once this was done, we benchmarked our program to identify potential bottlenecks. Even though the main bottlenecks accounted for only a tiny fraction of the total program execution time, we rewrote the associated codes because, as Amdahl's law dictates, we would not have been able to achieve large speedups on large distributed systems otherwise.



Methods

Lambda expressions

The `C++11` standard introduces lambda expressions which allow us to define anonymous function object (or functor). These are very convenient because they can be defined locally and passed as arguments to a function.

Furthermore, lambda expressions are capable of capturing variables in scope either by copy or reference. In a context where one starts with a sequential program and wishes to parallelize it in the form of tasks, this is extremely convenient because portions of the program such as the contents of a for loop can be inserted inside a lambda expression, thus forming a task.

Asynchronous task

The `hpx::async` function takes as arguments a function (a function pointer or a lambda expression) and its arguments. Its behavior is the same as the async defined in the standard `C++` library.

Let's suppose that the function passed as an argument to `hpx::async` returns a double type. `hpx::async` then returns a future of double, i.e. a `hpx::future<double>`. This future allows us to delay the execution of the function and to execute it on one of the available processors. As long as the execution of the function is not completed, the future suspends the execution of all `HPX` threads trying to access the result.

Simultaneous use of both features

The two features mentioned above have an extremely strong synergy. Indeed, we have seen that lambda expressions allow us to define tasks locally whereas `hpx::async` can execute them asynchronously. This is everything you need to parallelize a program.

But it is not that simple in practice and in the case of the FMM, the main troubles come from the nature of the algorithm.

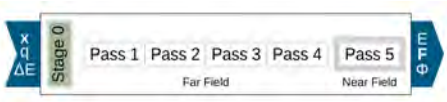


Figure 1: FMM Workflow

In its initial version, the program can be broken down into 5 passes. Passes 1 to 4 correspond to calculations of far field interactions, i.e. interactions between well separated particles. On the other hand pass 5 corresponds to the calculation of near field interactions, i.e. interactions between particles that are too close for the FMM to be used via the interactions of multipole expansions.

When executing this algorithm sequentially, it is sufficient to execute these passes one by one in the correct order (except for pass 5 which can be executed at any time). But what about once parallelized? Let's take a look at the two steps of pass 1 to understand the problem.

Figure 2: Pass 1 P2M

During the first part of pass 1, often referred as P2M for particule to multipole, we compute a multipole expansion for each box at the lowest level of our tree.

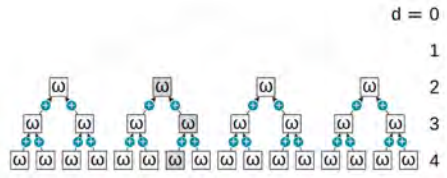


Figure 3: Pass 1 M2M

During the second part of pass 1, often referred as M2M for multipole to multipole, we combine those multipoles to obtain multipoles of groups of boxes.

As you can see, computing a multipole at depth $d = 3$ requires the prior calculation of two multipoles (because the scheme is for 1D, otherwise in 3D we would have 8) at depth $d = 4$. The same rule applies to all other levels. If we keep the original structure of the code which calculates the multipoles level by level, we are forced to insert synchronization barriers at each level to ensure that all underneath tasks are completed.

This is not ideal and some may prefer to use an operator like `hpx::when_all` so that each multipole calculation can begin once the tasks associated with the calculation of its two underneath multipoles are completed. The only problem is that this way of proceeding is very different from the sequential version and so we reach the first disillusionment: modifying a program to work with tasks is not trivial and often requires extensive code changes, making it impossible to simply wrap the loops of our program with lambda expressions. Fortunately, none of this is impossible!

Results

The holy pass 2

If we had to mention one thing that went particularly well, it would be the parallelization of pass 2.

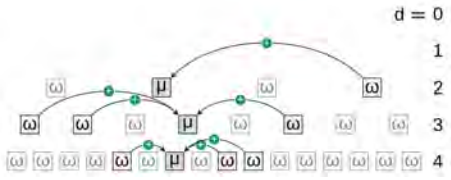


Figure 4: Pass 2

During this pass, we translate remote multipoles into local Taylor moments. As you can see, there is no particular difficulty because the involvement of multipoles in the calculation of those local moments is read-only! We can therefore create a task for the calculation of each moment without worrying about anything!

Let's have a look at the benchmark results for this pass with a number of particles n=12960, a depth d=4 and a multipole order p=20 as parameters. The benchmarking computer has 4 sockets, each containing one Intel(R) Xeon(R) CPU E7-4830 v4 @ 2.00GHz (14 cores per socket). This configuration

will remain the same in all the following graphs.

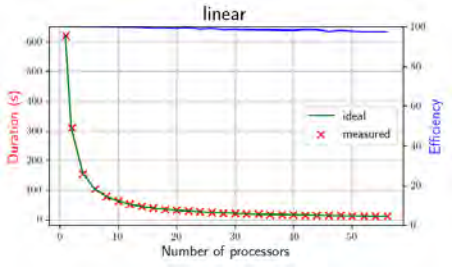


Figure 5: Pass 2 benchmark (linear scale)

At first sight, it seems that we almost have a linear speedup. The graph can be switched to logarithmic scale to see more detail.

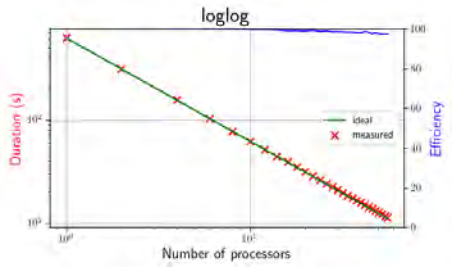


Figure 6: Pass 2 benchmark (logarithmic scale)

As you can see, with 56 processors the parallelization efficiency is 97%. As if that wasn't pleasant enough, this pass is responsible for most of the program's execution time (more than 95% in fact). As these results were good, it was time for us to optimize the rest of the code. Without knowing it, we were about to discover the 80-20 Rule.

The cursed passes

After this period of extraordinary results came a period of desolation during which the slightest acceleration was expensive, very expensive in terms of time and modifications compared to the sequential version. The smallest gains were those that took the longest to achieve.

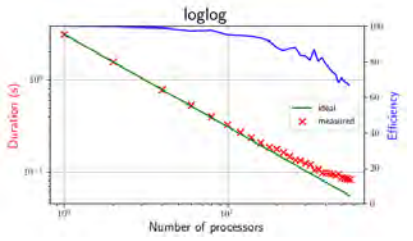


Figure 7: Pass 3 benchmark (logarithmic scale)

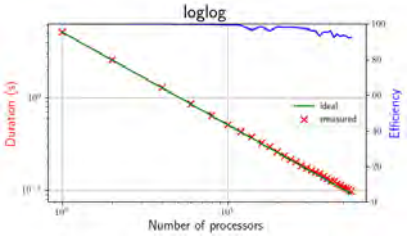


Figure 8: Pass 4 benchmark (logarithmic scale)

In the end, passes 3 and 4 were not that bad with efficiencies around 65 and 92% for 56 processors respectively. The results of passes 1 and 5 were less convincing. In fact, they were actually very bad. These are the main bottlenecks to be investigated in the future.

Conclusion

To conclude, let's add up the duration of each pass so we get the overall execution time.

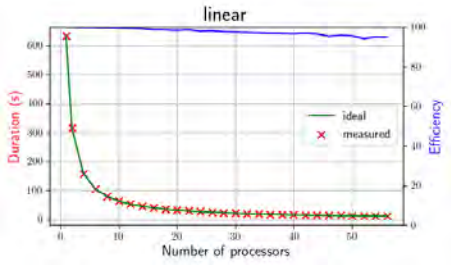


Figure 9: Full benchmark (linear scale)

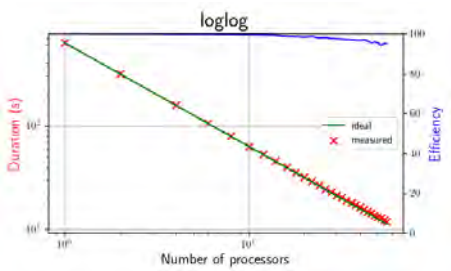


Figure 10: Full benchmark (logarithmic scale)

At first glance, one could say that the job is done. After all, the overall efficiency of parallelization is still 95% with 56 processors. If one or two passes are not perfect, it should not change much after all.

Let's assume that the two deficient passes are not modified and that their total duration is constant and equal to a percentage of the single thread execution time: let's say 0.1% (arbitrary). Then Amdahl's law will inevitably come back to haunt us. Let's do the math. We know that the theoritical speedup s is:

$$S = \frac{1}{f + \frac{1-f}{n}}$$

where:

- n is the number of processors
- f is the fraction of the problem that must be computed sequentially (in our example 0.1%)

This implies that:

$$S \leq \frac{1}{f}$$

In other words, the speedup will never exceed $1/0.001 = 1000$, no matter what resources we have available (it's something that money can't buy). This may sound extremely frustrating, but it tells us something very deep about high performance computing: this is a very merciless field that only rewards those who spend a lot of time and energy tracking down the smallest bottlenecks!

PRACE SoHPCProject Title

Tiny, tiny, tasks! Huge Impact?

PRACE SoHPCSite

Jülich Supercomputing Centre (JSC), Germany

PRACE SoHPCAuthors

Arthur Guillec, Tristan Michel, Polytech Sorbonne, Paris, FRANCE

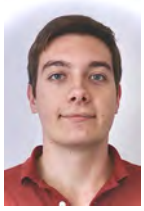

Arthur Guillec

PRACE SoHPCMentor

Ivo Kabadshow, JSC, Germany

PRACE SoHPCContact

Kabadshow, Ivo, Jülich Supercomputing Centre
Phone: +49 2461 61-8714
E-mail: i.kabadshow@fz-juelich.de


Tristan Michel

PRACE SoHPCSoftware applied

`HPX` created by the Stellar Group

PRACE SoHPCMore Information

www.stellar-group.org

PRACE SoHPCAcknowledgement

Our deepest thanks to Ivo for being an exceptional mentor, always available when we encountered difficulties.

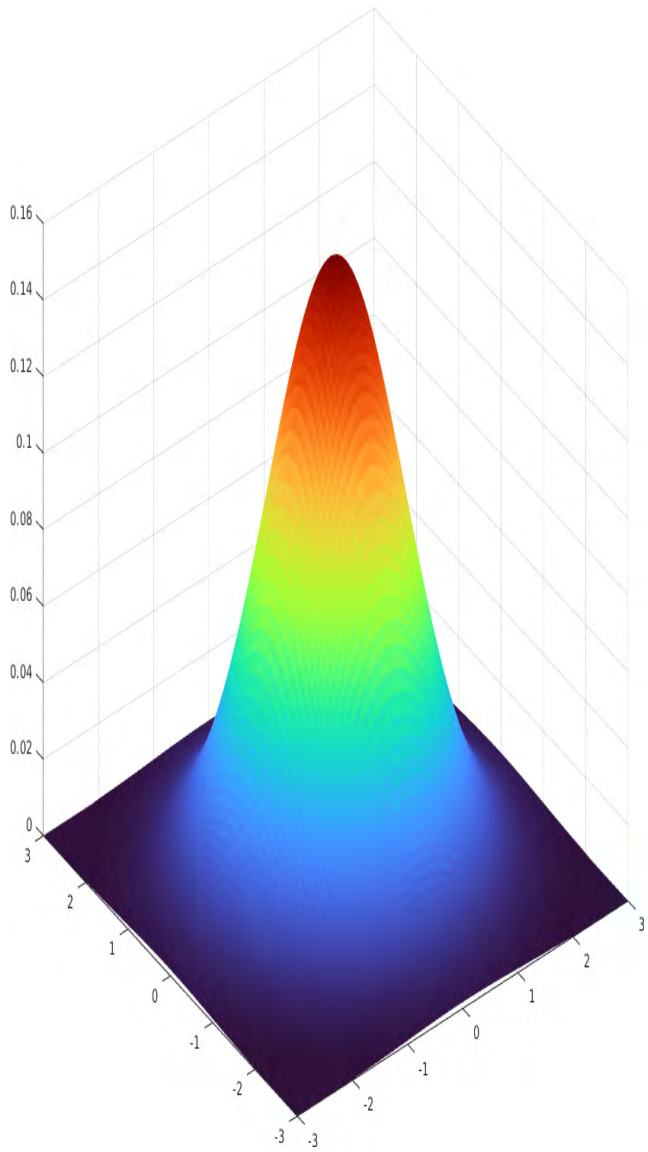
PRACE SoHPCProject ID

2121

Inhomogeneous equation with parallel computations of collisions

David Knapp and *Artem Mavliutov*

In order to accurately simulate the equation, it is important to introduce the transport part, thus making it a non-homogeneous equation. The main bottleneck of the equation is the computation of the collision term, therefore, the first step to obtain a fast solution is to parallelize the collision operator in an optimal way.



The Boltzmann-Nordheim equation or quantum Boltzmann equation describes the time evolution of a gas. When this gas formed by bosons is cooled down to a low temperature we can observe a Bose-Einstein condensate which is one the most striking quantum phenomena in nature. It is also described as the fifth state of matter. The study of this equation is still undergoing and can bring us closer to an understanding of this quantum effect.

We are interested in the following kinetic equation that represents the behavior of the distribution function $f(t,x,v) : \mathbb{R}_+ \times \Omega_x \times \mathbb{R}^{d_v} \to \mathbb{R}$ which characterizes the number of particles per unit volume $d_x \times d_v$ in the phase space $\mathbb{R}^{d_x} \times \mathbb{R}^{d_v}$ at time $t \in \mathbb{R}_+$:

$$\partial_t f + v \cdot \nabla_x f = c\, \mathcal{Q}(f)\,,$$

where $x \in \Omega_x \subset \mathbb{R}^{d_x}$ is the position,
$v \in \mathbb{R}^{d_v}$ is the velocity,
$t \in \mathbb{R}_+$ is the time variable,
$c \geq 0$ is a dimensionless scaling parameter for the collisions,
$\mathcal{Q}(f) : \mathbb{R}_+ \times \Omega_x \times \mathbb{R}^{d_v} \to \mathbb{R}$ is the collision operator.

Numerically this equation is solved by discretizing the time and solving separately the transport equation and subsequently solving the collision step with initial solution of the transport. In the next iteration the initial solution for the transport equation is taken as the solution of the collision equation in the previous time step.

Introducing the transport part, thus increasing the dimensionality of the problem by 2-3 dimensions in space is one of the goals of the project. The other goal deals with the optimization of parallel computation of the collision kernel.

The collision kernel of the simulation is computed using spectral methods.[1] The collision term of the equation can be split into six parts, where four of them are used to describe the behavior of the quantum particles. The work of David focuses on the first of the quantum collision terms, which takes most of the time during the computation.

$$Q_{1,q} = \int_{\mathbb{R}^{d_v}} \int_{\mathbb{S}^{d_v-1}} B\left(|\xi - \xi_*|, \theta\right) G(t,x,\xi) G(t,x,\xi_*) G(t,x,\xi_*) d\sigma d\xi_*$$

The goal was to implement a new elaborate method using advanced MPI techniques to decrease the work each process has to do during the computation of $Q_{1,q}$.

Methods

Transport part

The transport part is solved with the Finite Volume Method.[2] The principal

motivation for this method is its conservational property which is the main key when modeling physical process. Therefore, the conservation of flux is done by integrating the distribution function in space:

$$\int_{x_{i-1}}^{x_{i+1}} f(t^{n+1},x)dx = \int_{x_{i-1}^t}^{x_{i+1}^t} f(t^n,x)dx,$$

where $x_{i\pm1}^t = x_{i\pm1} - v\Delta t$.
In order to correctly implement a conservational method, we have to rewrite this equation in an updating form. Thus, setting

$$\Phi_{i+1}(t^n) = \int_{x_{i+1}-v\Delta t}^{x_{i+1}} f(t^n,x)dx$$

the conservational form becomes:

$$\int_{x_{i-1}}^{x_{i+1}} f(t^{n+1},x)dx = \int_{x_{i-1/2}}^{x_{i+1}} f(t^n,x)dx + \Phi_{i-1}(t^n) - \Phi_{i+1}(t^n)$$

The next step is the reconstruction of the primitive function on each sub interval $[x_{i-1}, x_{i+1}]$ using Lagrange interpolation:
1. Compute the average

$$f_i^n = \int_{x_{i-1}}^{x_{i+1}} f(t^n,x)dx$$

$$F(t^n,x_{i+1}) - F(t^n,x_{i-1}) = \Delta x\, f_i^n$$

2. Interpolate the primitive function

$$\tilde{F}_h(t^n,x) = F(t^n,x_{i-1}) + (x-x_{i-1})f_i^n$$

3. Differentiate the primitive function to obtain the distribution

$$\tilde{f}_h(t^n,x) = \frac{\partial \tilde{F}_h(t^n,x)}{\partial x}$$

Collision part

The computation of the $Q_{1,q}$-collision term can be improved by using advanced MPI-communication developed by Alexandre Mouton. Implementing the new MPI-communication leads to two adaptation in the current state of the algorithm. First, we use a custom communicator in the computation of $Q_{1,q}$where we have to organize the splitting of the processes of the old `MPI_COMM_WORLD` communicator. Second, we have to adapt the distribution of the data over the processes, such that every process in every communicator has the data it needs to fulfill its computation.

We assume, that we have $S \times \tilde{S}$ processes in the `MPI_COMM_WORLD` communicator. For our pattern of communicators, we want to split the `MPI_COMM_WORLD` communicator in two ways, once along the rows and once along the columns. For the communicator along the rows, this results in $\tilde{S}$ communicators with S processes each.

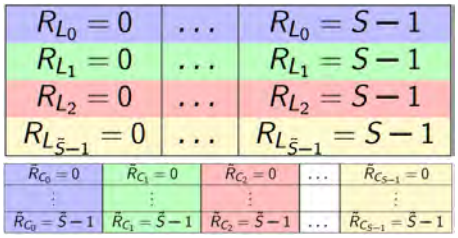


Figure 2: The splitting of the `MPI_COMM_WORLD` communicator. At the top the splitting along the rows, at the bottom, the splitting along the columns.

For distributing the data to the processes we take advantage of the fact, that the process of rank 0 in the first row communicator is also the process of rank 0 of the column communicator. In our case, data of the i-th process in the first row communicator is needed on every process in i-th column communicator. With this communicator-structure we can use `MPI_Bcast` on every column-communicator to distribute the data.

Compared to the original version of the computation of $Q_{1,q}$we need more communication. But due to the communication we can avoid a a large loop over the whole computational domain, which should decrease the computational cost on every process

Initial and boundary conditions

The boundary conditions are periodic in the space domain $x \in \Omega_x$ and in the velocity domain[3] $v \in [-L,L]^{d_v}$.

The initial distribution is the following:

$$f^0(x,v) := g(x)\tau(v)$$
$$x \in [0,1]^{d_x},\ v \in [-L,L]^{d_v},$$
$$g(x) := 1 + 0.5\cos(2\pi x)$$
$$g(x) \geq 0,\ x \in [0,1]^{d_x},$$

where τ is the classical Maxwellian distribution.

Results

Transport part

Figure 1 shows the time evolution of the distribution function which was computed in 5 dimensions (1D time, 2D space, and 2D velocity). Unfortunately, only 1st order accuracy is guaranteed. To alleviate this problem, it is necessary to introduce slope correctors[4] that limit the high order terms at the points in which the gradients are steep. Only then it is possible to introduce high order interpolation terms. Another problem regarding accuracy is the size of the mesh. Increasing the mesh size improves the accuracy but also the computational time. Finally, we have to be careful when choosing the size of the velocity domain since we must fulfill the CFL[1] conditions.

Collision part

After implementing the method, we investigate the scaling of it and compared it with the original implementation of the $Q_{1,q}$collision term. We compared the two implementations using two experiments. For the first one, we run the simulation using eight processes and increased the size of the grid in every new run. Initially we use a 16×16 grid. After each run, we doubled the size of an edge of the grid. In the second experiment, we fixed the size of the grid to 64×64 and increased the number of processes after each run, starting with a run using two processes.

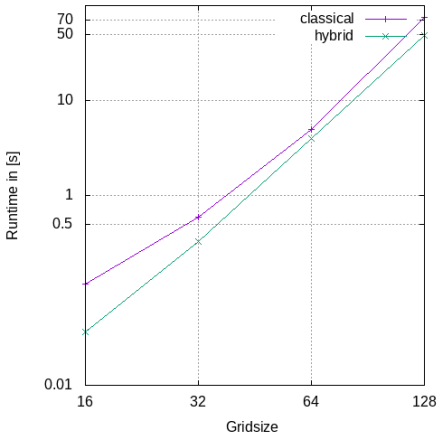


Figure 3: The results of running the simulation with 8 processes and different grid sizes. The graph hybrid describes the results of the new implementation, classical the original version

Figure 3 shows, that the new implementation is faster than the original one. Nevertheless, the new method scales similar to the original implementation regarding the size of the grid. A reason

[1] Courant–Friedrichs–Lewy condition: $C = \frac{v\Delta t}{\Delta x} \leq C_{max}$ where C_{max} is usually taken as 1 for the explicit time marching scheme. This condition is necessary for the convergence of a numerical scheme.

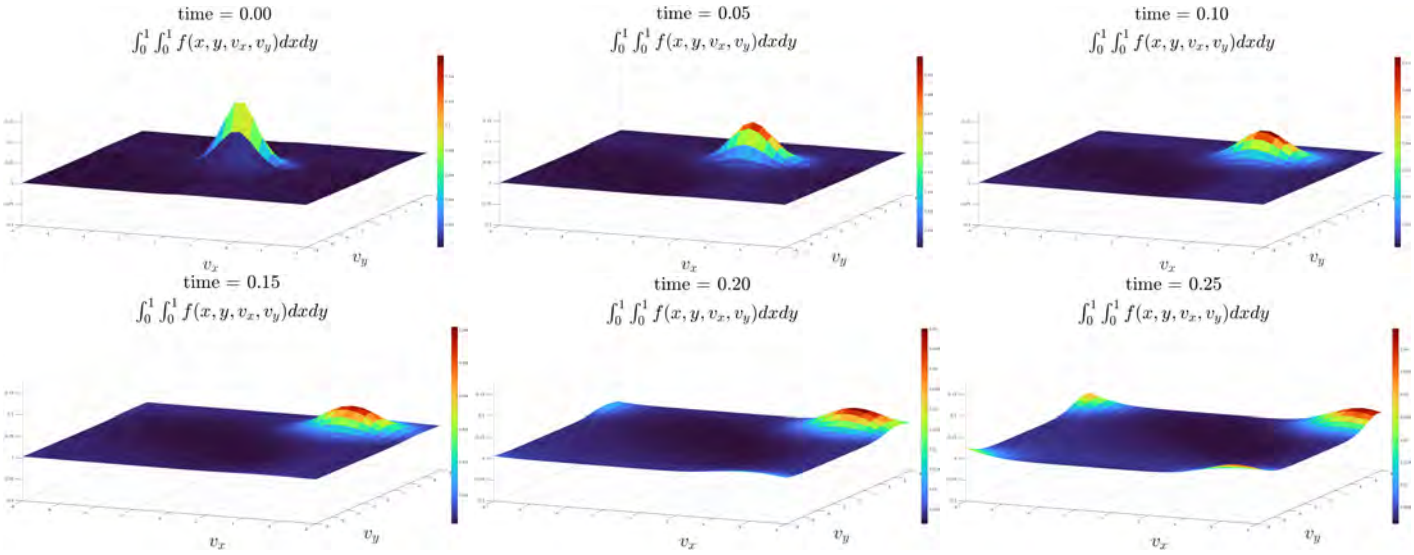


Figure 1: Snapshots of the time evolution of the distribution function $f(t, x, v) : \mathbb{R}_+ \times \Omega_x^{2D} \times \Omega_v^{2D} \to \mathbb{R}$ integrated in space

for this behavior can be, that with an increased size of the grid, the communication takes more time. Nevertheless, the remaining part of the computation if fast enough, such that the new method remains faster.

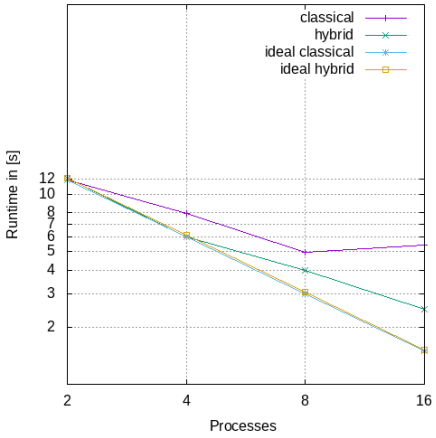


Figure 4: A comparison of the scaling of the classical and the hybrid implementation

Figure 4 shows again, that the new implementation is faster than the original one, but we can also see, that the hybrid method scales better than the original one. It is interesting to observe, that the two methods have a similar runtime for two processes. But the hybrid method scales faster and for more processes.

Discussion & Conclusion

Transport part

It is safe to say that the initial version of non-homogeneous Boltzmann-Nordheim equation was implemented successfully. Moreover, we were able to increase the space dimension to 2D. The future work can be focused primarily on improving the accuracy and handling of numerical instabilities. Furthermore, extension to 3D case in space can be seen in the future. Also, for the correct solver to work it is absolutely necessary to parallelize even further the algorithm keeping in mind the specificity of non-homogeneous equation. Finally, the current version of the transport is parallelized with `openMP` later it can be further adjusted to `MPI` and `CUDA`.

Collision part

Two month ago, we had the goal to implement a better scaling and faster version of the computation of the $Q_{1,q}$term in the computation of the quantum collision in the Boltzmann-Nordheim equation. With the presented results we delivered such implementation. Nevertheless, we can improve the computation even further. The spectral accuracy of the current implementation can be enhanced, as we observe errors at the magnitude of $10e^{-8}$, but we should be able to decrease it down to machine precision. Additionally we should testify the current results with more processes and on a larger grid-size as both the number of used processes and the size of the grid are not very large. We observed, that the communication between the processes is more intensive in our version of the code. If we can decrease the communicational effort without changing the overall method, we could improve our implementation even further.

Acknowledgments

We would like to thank our mentors Alexandre Mouton and Thomas Rey for giving us the opportunity to work on this interesting project. Thank you, for supporting us during our internship and for giving us an insight into the topic. We also thank the Laboratoire Paul Painlevé, which supports the KINEBEC project. Furthermore we want to thank the coordinators of the Summer of HPC 2021 for the opportunity to be a part of this project.

References

[1] Francis Filbet, Jingwei Hu, and Shi Jin. A numerical scheme for the quantum boltzmann equation with stiff collision terms. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 46(2):443–463, 2012.

[2] Russo G. Filbet F. *Modeling and Simulation in Science, Engineering and Technology*. Birkhäuser, Boston, MA, 2004.

[3] Lorenzo Pareschi and Giovanni Russo. Numerical solution of the boltzmann equation i: Spectrally accurate approximation of the collision operator. *SIAM Journal on Numerical Analysis*, 37(4):1217–1245, 2000.

[4] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

PRACE SoHPCKINEBEC
Numerical simulation of Boltzmann-Nordheim equation

PRACE SoHPCLille, France
Laboratoire Paul Painleve, France

PRACE SoHPCAuthors
David Knapp,
University of Bonn, Germany
Artem Mavliutov,
University of Padova, Italy

PRACE SoHPCMentor
Alexandre Mouton, Lille, France

PRACE SoHPCContact
E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied
KINEBEC

PRACE SoHPCMore Information
KINEBEC

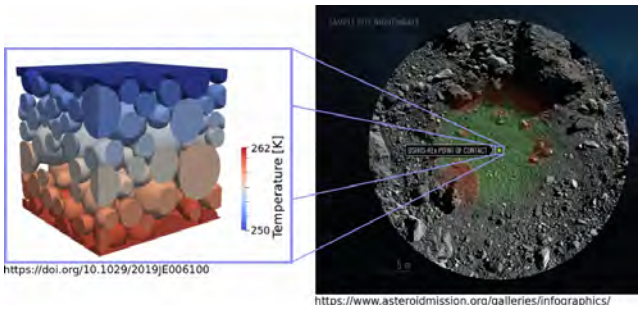
PRACE SoHPCAcknowledgement
Thanks to Alexandre Mouton and Thomas Rey

PRACE SoHPCProject ID
2122


David Knapp


Artem Mavliutov

Parallel radiative heat-exchange solver



Cormac McKinstry, Venkata Mukund Kashyap Yedunuthala

Develop a parallelized radiative heat exchange solver that can simulate the thermal activity of regolith samples collected by the OSIRIS-REx space mission

Abstract

Objective of the project is to develop a Finite Element Method based solver for radiative heat exchange observed in the material present on the surface of asteroids, termed *Regolith*. Solution of heat transfer equation accounting for radiation requires evaluation of view factors between any two elements within the meshed domain. Computationally expensive evaluation of double area integral to determine view factors requires evaluation of the "visibility" between such elements, which is done through implementation of ray-tracing libraries. The view factor integral is approximated to algebraic equations through Gaussian Quadrature. Upon successful evaluation of approximated values of view factors through application of quadrature, the view factors are integrated into the evaluation of weak-formulation of heat transfer equation. Thermal response is to be estimated then through Finite Element Method. Parallelisation is to be implemented through libraries such as PETSc, FEMS. These libraries support shared memory and distributed memory development.

Introduction

On the 8th of September 2016, NASA launched the OSIRIS-REx space exploration mission from Cape Canaveral. It launched with the aim of landing on the near-Earth asteroid 101955 Bennu, collecting samples from it's surface and transporting them back to Earth for analysis.

The spacecraft landed on Bennu on the 20th of October 2020, and successfully gathered between 400g and 1kg of regolith, the loose, granular rocky material that covers the surface of Bennu and many asteroids like it. The spacecraft is due to arrive back on Earth in September 2023.[2]

The purpose of this project is to study the thermal responses of this regolith. We want to understand it's behavior under different thermal situations, and how they vary with factors such as the sizes of the particles. We want to know this as asteroids such as Bennu can act as "time capsules", indicating what might have been seen around the formation of the Solar system.

Sections of the project

This was a project of two halves

1. The calculations of the view factors of a given arrangement. These are the coefficients necessary for to determine thermal radiation. This was done by Cormac McKinstry
2. Upon calculation of view factors, further task was to simulate the thermal behaviour by calculating temperatures through finite element method. This part was handled by Venkata Mukund Kashyap Yedunuthala.

View Factor Calculation

The view factor is a coefficient describing how thermal radiation moves between surfaces. It's calculation and accuracy is vital to the heat exchange solver, but doing so in reasonable time accurately is a non-trivial challenge.

Theory

The view factor F_{12} describes, of the thermal radiation radiated by Surface

1, how much of it is received by surface 2. The view factor is described by the equation,

$$F_{12} = \frac{1}{A_1} \int_{A_1} \int_{A_2} \frac{Cos\theta_1 Cos\theta_2}{\pi r_{12}^2} dA_1 dA_2 \quad (1)$$

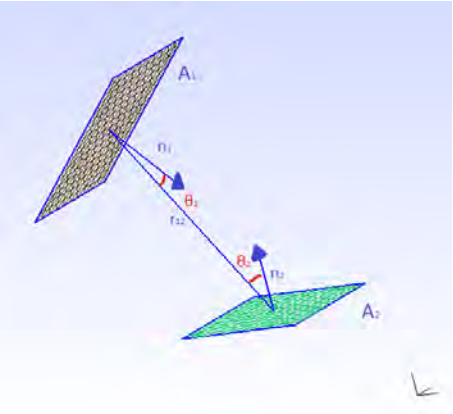


Figure 1: Two surfaces that see one another

The double area integral is computationally expensive to do accurately, and benefits from all speed increases that can be found. That was the crux of the problem.

The arrangement of regolith was described by a surface mesh of a set of non-touching spheres. The first question that needs to be answered is what surfaces have a line of sight from one to the other. For this we use Embree.

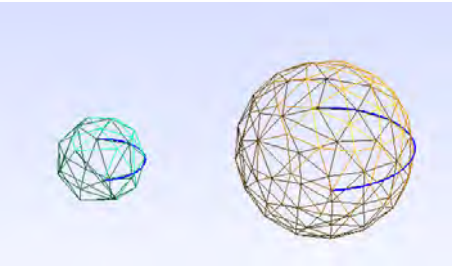


Figure 2: A mesh of triangles representing two spheres

Embree

Embree is Intel's ray tracing package and collection of kernels. While Embree is a large and multifaceted package including many advanced ray casting and rendering functions, we simply use it to determine whether there is a free line of sight between two triangles. We use Embree due to it's versatility across platforms, and it's inbuilt optimization and parallelisation strategies.[3]

Integration methods

The choice of integration methods for the double area integral was central to the view factor problem. For this project gaussian quadrature methods were implemented, choosing particularly chosen weighted points such that they returned accuracy that would require multiple orders of magnitude more points to be selected to achieve with evenly weighted points. The implementation was set up so that at use the program could be set to sample 1, 6, 16 or 64 points, depending on what constituted acceptable run-time and accuracy for the user. This was important as for n sampled points, n^2 calculations were to be done.

Parallelisation

We sought to parallelize the view factor calculation process in order to speed up the process. After the sequential version of the code was tested, a version implementing parallelisation with MPI was made.

Currently, the implementation of parallelisation is a simple, naive one, where each processor is assigned a set of triangle-pairs to check if the see one another and if so calculate the view factor. This is unoptimal, as it will likely have non-balanced processor workloads, with some dealing with many pairs of triangles that see one another and must be integrated over, and some having very few or even none. There are further steps that could in future be taken to balance the processor workload and speed up the computation time.

Finite Element Method

Introduction

Finite element method is a numerical technique developed in the past few decades. It is primarily used to solve boundary value problems through division of a domain into a number of sub-domains, also known as elements. A boundary value problem is the combination of an initial value problem, which in turn could be a set of ordinary or partial differential equations, and the boundary conditions of the domain. Boundary conditions reflect the conditions of the surroundings of the domain in consideration. The solutions are obtained on the nodes of an element, which are the points that make up the corners of an element. Typically, the boundary value problem is converted to a set of algebraic equations through different mathematical manipulations, and such a set of equations is solved iteratively for each element for a given period of time. The solutions are obtained in an assembled form at the end of each time step. This procedure within the context of this project are explained below.

Theory

Heat transfer could be described as the process of flow of heat between two bodies. Considering that regolith is a body in space, the process of relevant heat transfer could be modeled using following equation:

$$\rho c \dot{T} + q_{i,i} - f = 0 \quad (2)$$

Here in equation (2), $q_{i,i}$ describes heat transfer contribution through conduction and radiation, with ρ being density of the material, c being the specific heat capacity of the material, and f refers to source terms. Additionally, reducing the equation (2) to its so-called weak-formulation is necessary for FEM formulation. Contributions from conduction and radiation are modeled using Fourier's law (3) and Stefan-Boltzmann law (4), which are described as follows:

$$q_i = -kT_{,i} \quad (3)$$

$$q_i = \epsilon\sigma \sum_{j=0}^{N} F_{ij}(T_i^4 - T_j^4) \quad (4)$$

In equation (3), the term k refers to thermal conductivity of the material. The Stefan-Boltzmann law, as described in (4) is of particular importance in this project. It is important to note that in this particular formulation, the radiation terms act as a boundary condition, since the radiation is mostly from the Sun and is observed from a single direction. In equation (4), the term F_{ij} describes the view factors between two elements termed i and j, the term ϵ refers to emissivity of the material. The term σ is known as Stefan-Boltzmann constant and has the value of $5.670374419 \times 10^{-8} Wm^{-2}K^{-4}$. The view factors, evaluated as described in the earlier section, are integrated into this particular problem at this point.

Upon evaluation of weak-formulation as described above, the

terms are converted into matrix formulations suitable for finite element method. This procedure is known as discretization. The resultant algebraic equations are now solved iteratively. This is achieved through usage of *FEMS* – a finite element computing software developed by Modesar Shakoor, our co-mentor.

Meshing

Meshing is the process that divides a given domain into elements, as required by FEM computations. FEMS offers its own tools to create such meshes, and offers interfaces to import meshes that were created by third-party softwares like *GMsh*. As opposed to view factor computations, FEM computations are done on a volumetric mesh. As such, this requires appropriate correlation of elements present on the surface mesh with elements present in volumetric mesh. In this project, surface mesh was constructed using triangular elements and volumetric mesh was constructed using tetragonal elements. Therefore, a comparison of barycenters is thought to be an appropriate method for such a correlation. As the project is divided into two halves, this required an efficient file handling to export and import view factors for corresponding elements. An example mesh could be seen in Fig.3.

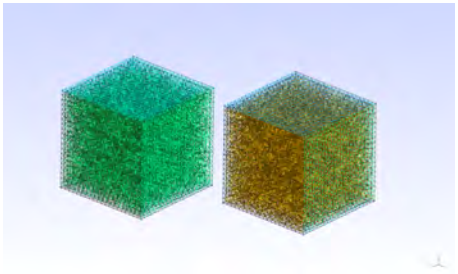


Figure 3: Example volumetric mesh for two cube-shaped domains

FEMS

FEMS as a software is developed entirely in C. It uses libraries such as *LibXml2* for XML-document parsing, which are used to configure a certain problem, and *GLib2* for additional data structures. The software is compiled using *CMake*, which offers high flexibility for inclusion of any additional libraries to enhance the functionality of this software package. As one can imagine, the operations as described above could be, computationally, highly expensive. Fundamentally, FEMS offers shared memory parallel implementations using *OpenMP* at its foundations, as well as linear algebra libraries such as *CBLAS/LAPACKE* or *Eigen3*. Additionally, for some components, FEMS enables parallelization through *PETSc* and *MPI*. This project was to be built upon such an implementation of FEMS.

PETSc

PETSc – the Portable, Extensible Toolkit for Scientific Computation, provides data structures that enable seamless parallelization of a scientific computing problem. It is vital in this project as the system of algebraic equations obtained as described above conduces to a sparse system that is different to classical Finite Element Method. Therefore, one needs to provide or alter the existing sparsity pattern in order to correctly account for resultant contributions from radiation. PETSc affords the flexibility to achieve this particular aspect.

Conclusions and discussion

In this study, we had focused on a specific case of heat transfer that is of particular importance to space exploration and astrophysics.
As discussed, the complicated mathematical modeling and computation of such a convoluted physical phenomenon involves a high degree of complexity. Advancements in mathematics to solve differential or integral equations, such as quadrature or time integration schemes assume utmost importance in such an analysis. Implementation of a parallel Finite Element Method solver is expected to significantly ease such an analysis.

Appendix

Project Timeline

Week 3 (5 July – 25 July)

Read the necessessary bibliography to understand the situation and the problem. Installed and set up the relevent pieces of software, libraries, etc.

Week 4: (26 July – 1 August)

Initial programming of the view factor calculations and the nonlinear finite element solver using PETSc.

Week 5 (2 August – 08 August)

Continue work on programming, gather initial results.

Week 6 (08 August – 15 August)

Refinement, correction and fixing of code, reassessing choices to gain more accurate results.

Week 7 (16 August – 22 August)

Validation of results, comparing results to existing knowledge.

Week 8 (23 August – 31 August)

Parallelized the existing code for greater efficiency, prepared final reports.

References

[1] Shakoor, Modesar. (2021) FEMS – A Mechanics-oriented Finite Element Modeling Software. *Computer Physics Communications*, 260:107729.

[2] Wall, Mike (October 2020) NASA's OSIRIS-REx Probe Successfully Stows Space-Rock Sample, Scientific American

[3] Áfra, Attila & Wald, Ingo & Benthin, Carsten & Woop, Sven. (2016). Embree ray tracing kernels: overview and new features. 1-2. 10.1145/2897839.2927450.

PRACE SoHPC Project Title

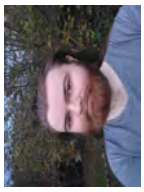
Parallel radiative heat-exchange solver for analyzing samples from the OSIRIS-REx space exploration mission

PRACE SoHPC Site

MdlS-Maison de la Simulation (CEA/CNRS), France

PRACE SoHPC Authors

Cormac McKinstry, Venkata Mukund Kashyap Yedunuthala



Cormac McKinstry

Venkata Mukund Kashyap Yedunuthala

PRACE SoHPC Mentors

Daniel Pino Muñoz, MINES ParisTech, France.
Modesar Shakoor, IMT Lille Douai, France

PRACE SoHPC Contact

Cormac, McKinstry, Ireland.
E-mail: mckinstc@tcd.ie
Venkata Mukund Kashyap, Yedunuthala, Germany.
E-mail: mukund.yedunuthala@student.tu-freiberg.de

PRACE SoHPC Software applied

Embree, FEMS,[1] PETSc

PRACE SoHPC More Information

https://www.embree.org/
https://www.mcs.anl.gov/petsc/
https://doi.org/10.1016/j.cpc.2020.107729

PRACE SoHPC Acknowledgement

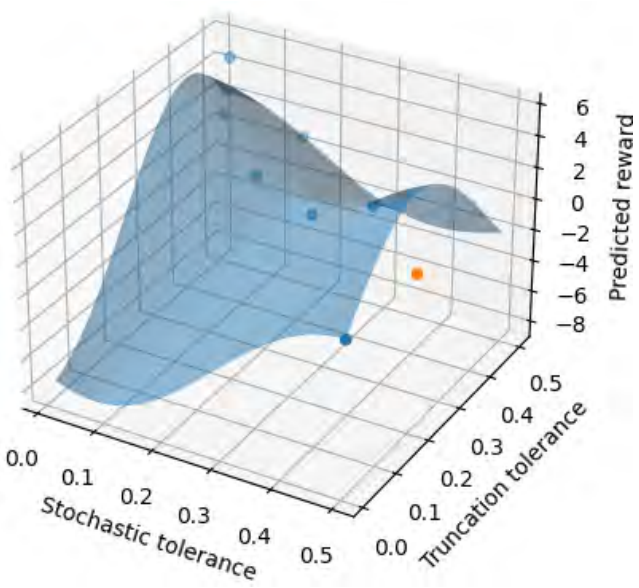
We thank our mentors, Daniel Pino Muñoz and Modesar Shakoor for their co-operation and support. We are also grateful for the Summer of HPC initiative by PRACE.

PRACE SoHPC Project ID

2123

Bayesian parameter search for matrix inversion

Adrian Lundell and Iakov Kharitonov



We implement a parameter search for an advanced matrix inversion algorithm to accomplish faster solving speeds. Furthermore, we perform benchmarking and analysis to find improvements and extensions for increased usability.

Introduction

Solving of large systems of linear equations, or specifically finding the inverted matrix A^{-1} given by

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

is of great importance in a wide range of fields, from economics to engineering. One way to increase the computation speed is to multiply the equation with the *preconditioner* matrix M to get an easier equation system

$$MA\mathbf{x} = M\mathbf{b}. \tag{2}$$

No general method of computing a good preconditioner exists but there are a number of algorithms for creating approximate solutions, typically parameterized with a number of values heavily impacting their effectiveness.

With this project we aim to develop an efficient parameter search algorithm for the Markov Chain Monte Carlo for Matrix Inversion (*MCMCMI*) approximation using a Bayesian sampling approach. We also look into the possibilities to use information from previous preconditioner computations by studying the correlations between a set of matrix features and their optimal parameters.

Theory

The motivation of **Bayesian inference** in context of the problem presented is to gain insight into the posterior distribution over some unknown parameters of the model. Posterior distribution is the conditional distribution for variables that haven't been observed yet given the data (which has been observed). For the majority of cases, it is not deemed possible to analytically derive it. Luckily, for the Bayesian sampling approach, knowing it is not required. We can construct methods that sample exactly from the posterior distribution and use statistics from the samples to represent that distribution. Using these statistics, we can make inferences and predictions.
Our algorithm can be generally outlined in the following way:

1. It takes the data (previously made samples) and builds a regression model from it
2. Using the method described in detail below, it computes the most optimal set of model parameters
3. Guided by the best parameters, it takes a new sample (evaluates the

model) with a strategy that tries to minimise biased sampling

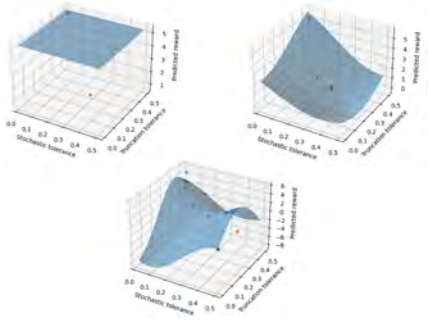


Figure 1: Three steps of the algorithm showing the convergence of the model.

As shown by figure 1, this is an iterative process that aims to improve on the quality of inferences based on the modelled distribution as it progresses. It experiences certain pitfalls, which will be further discussed along with attempts to avoid them.

Markov Chain Monte Carlo refers to a class of algorithms based on first constructing a process to generate samples with a probability distribution mirroring the problem to be solved, followed by sampling this process to find said distribution. For the case of matrix inversion the algorithm was first proposed by Neumann and Ulam, distributed in print by MTAC,[1] and later on extended to general matrices and implemented in C++ and Python for large architectures by Lebedev, A., Alexandrov, V. and Strassburg, J.[23] The algorithm works by performing a number of random walks on the matrix and iteratively updating a sum for each matrix element every time it is reached, choosing the path and calculations such that the expected average value for each element turns out to be the sought after inverse. This introduces one parameter ϵ or the stochastic tolerance determining the number of random walks, and one parameter δor the truncation tolerance, determining how the path is determined. In theory these could be set infinitely small to end up with the exact solution in infinite time, but for the computation of a preconditioner a calibration is necessary. For a detailed explanation of the algorithm please refer to.[3]

Gaussian process (GP) is a type of stochastic process that constitutes an infinite collection of random variables, where a finite selection of those is distributed according to a joint Gaussian distribution. That Gaussian distribution is characterised by its mean and covariance (kernel). Given a dataset (matrix evaluations by MCMCMI method), the Gaussian process defines a range of best fitting functions. The function of choice is the one that offers the best-behaved nonlinear fit to the data. This is Gaussian process regression.
The optimisation using GP is centred on computation of the expected gradient over the posterior distribution. f_* is the function evaluated at x_* and given the data X and the model $y = f(x) + \epsilon$ where ϵ is the Gaussian noise, its expected gradient[4] is

$$\nabla E|f_*|X, y, x_*| = \sum_{i=1}^{n} \alpha \nabla k(x_*, x_i) \tag{3}$$

where $\alpha = (K + \sigma_n^2 I)^{-1} y$.

I is the identity matrix and K defines the covariance for all datapoints, $K_{ij} = k(x_i, x_j)$. The RBF kernel is defined as

$$k(x_i, x_j) = e^{-\frac{d(x_i, x_j)^2}{2l^2}} \tag{4}$$

where l is the length scale and d is the Euclidean distance. Its gradient is

$$\nabla k(x_*, x_i) = k(x_*, x_i) \frac{x_i - x_*}{l^2} \tag{5}$$

As part of this optimisation approach, the calculated gradient makes it visually intuitive to understand the shape of the function, and the behaviour of the system in response to change in parameters. In the current case, as it will be explained later, we are considering a 3D surface because we are optimising two parameters.

Matrix features, such as norm, eigenvalues, ratio of non-zero entries to all entries (sparsity), symmetry, etc, are distinct characteristics of matrices and also means to classify them. In the context of initial motivation of this project, these features would be considered synonymous with “states” in ML algorithms.

Method

The algorithm was developed in *Python 3.9.5*, using the libraries *Numpy* and *Pandas* for data handling and the *GaussianProcessRegressor* from *sci-kit* learn for the Gaussian process regression. The Gaussian process was initialised with the standard RBF kernel and a limited length scale range between 0.3 and 10, as smaller length scales tended to favour an overfitted function surface. Furthermore a noise level of $\alpha = 0.001$ was introduced to prevent this issue.

To find the optimal parameter values for the Gaussian process we used gradient ascent with a fixed step length to find local maximas, repeating the process a number of times to ensure that the global maxima is found. Since the expected gradient of the Gaussian process is cheap to compute compared to each point evaluation, and since the model in itself is approximate making an exact solution unnecessary, this part of the algorithm was not deemed important to optimise further at this stage.

For evaluation, a previous Python implementation of the serial MCMCMI algorithm was used and the matrix was inverted with and without preconditioner to find the ratio between these two execution times and see the improvement in each sample. Using this relative reward value rather than some absolute timing allows for easier comparison between matrices of different complexity and matrices computed with different resources.

Initially in the project the goal was to include the matrix features into the statistical model to create a general model for all possible matrices, or at least a subset of a certain type. Afterwards it was, however, realised that the project was not yet mature for such an ambitious undertaking,and we had to separate the project into finding optimal points for single matrices, and studying the correlations between a number of matrix features and their optimal parameters.

This approach led to two types of experiments: first optimising parameters for single matrices, followed by plotting the optimal parameters as a function of their matrix features.

As far as the authors are aware this project is the first parameter search algorithm aimed at the MCMCMI algorithm making a qualitative benefit analysis difficult, instead we provide some overview of what performance can be expected and some lessons learned during the project on how to

improve the results going forward.

After initial development all experiments were run on the Hartree supercluster using *Python 3.6.5*, using a sample set of six sufficiently large matrices.

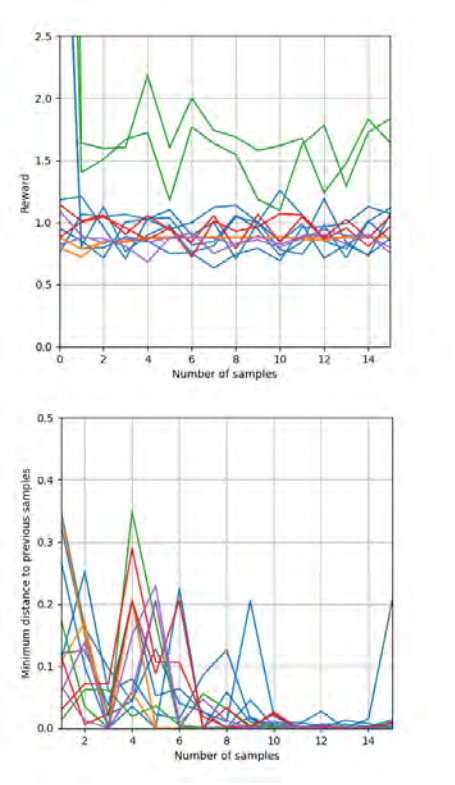


Figure 2: Convergence in reward and step length of the algorithm. Lines with the same color indicate several runs of the same matrix.

Results

A combined plot of the convergence of the reward and the parameter step length between samples for 6 different matrices can be seen in figure 2. It is apparent that the algorithm converges toward some set of parameters within 10 samples for all runs, but that these parameters are either not particularly optimal, or that the variance in computation time for the same parameter choice is similar to the variance from choosing parameters.
To find why this is the case, individual runs were studied in detail by plotting the model surface with more and more samples added. This resulted in three findings:

1. *Outliers*. For some samples there is a scale factor of five reward increase compared to samples very close in parameter space, raising suspicions that the result is due to externalities such as some load changes in the computer cluster node rather than the parameter choice.
2. *Heavy bias toward initial samples*. By design the algorithm tries regression with very few points, which in some cases causes it to get stuck in a local maxima of the initial samples. As with all search algorithms this is a question of exploration vs. exploitation, where it seems like the algorithm currently leans too much towards the latter option.
3. *Smooth reward surface*. As more points are added the Gaussian process tends to smoothen out rather than get more detailed, which sometimes causes maxima to be erased in favour of a cluster of semi optimal points close together.

Discussion

From the results it is clear that the algorithm may be improved with inspiration from the problems found in the testing data.

Outliers should be removed, with the slight complication of determining which points are outliers.

Naively one could start the algorithm with a rough grid sampling to avoid the problem of the exploration. A more sophisticated solution would be to borrow something close to the epsilon parameter in the epsilon-greedy algorithm from reinforcement learning within the optimiser method rather than finding the maxima all the time, allowing to decrease the exploration over time. To combat the smoothing a more in depth analysis of what type of surface to expect would be required, essentially finding kernels and parameters optimised for the matrices. For example the *Matern* kernel has a rougher surface which could suit the problem better.

One step towards a more generalised version of the algorithm would require the consideration of matrix features, defined previously. In that case, the model would extend over all the dataset (matrices) and not be confined to a single trial set of samples. This could help to get a general sense of behaviour of systems with features (e.g. matrix norms) close to ones evaluated and infer optimal regions in parameter space for similar classes of matrices. Such improvement would, of course, require a more extensive dataset with larger matrices. That, in turn, may lead to a practical necessity for parallelising the optimisation code and adopting the parallel version of MCMCMI due to increasing computation time.

As highlighted in the issue with outliers, using ratio of runtimes as the reward might be a biased approach. Large scale computing systems are likely to exhibit temporal load variability, even when working on a node exclusive from other users. This is due to scheduled tasks running in the background at different times of day that are generally beyond user's control. That leads to the idea of using the number of steps of MCMCMI method as the reward in order to exclude external factors. The downside is that the model would contain less information of several types of computation involved.

References

[1] Forsythe, G.E and Leibler, R.A. Matrix Inversion by a Monte Carlo Method

[2] Lebedev, A. and Alexandrov, V. On Advanced Monte Carlo Methods for Linear Algebra on Advanced Accelerator Architectures

[3] Strassburg, J and Alexandrov, V. On Scalability Behaviour of Monte Carlo Sparse Approximate Inverse for Matrix Computations.

[4] Rasmussen, C. and Williams, C. Gaussian Processes for Machine Learning

PRACE SoHPCProject Title

Hybrid AI Enhanced Monte Carlo Methods for Matrix Computation on Advanced Architectures

PRACE SoHPCSite

Hartree Centre, UK

PRACE SoHPCAuthors

Lundell, Adrian, Sweden
Kharitonov, Iakov, Russia

PRACE SoHPCMentor

Alexandrov, Vassil, UK
Lebedev, Anton, Germany

PRACE SoHPCContact

Adrian Lundell
E-mail: adrian.lundell@gmail.com
Iakov Kharitonov
E-mail: ik2318@ic.ac.uk

PRACE SoHPCSoftware applied

Python

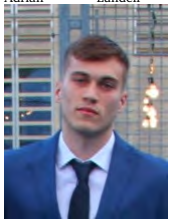
PRACE SoHPCAcknowledgement

Thanks to the Hartree Centre and our mentors.

PRACE SoHPCProject ID

2124

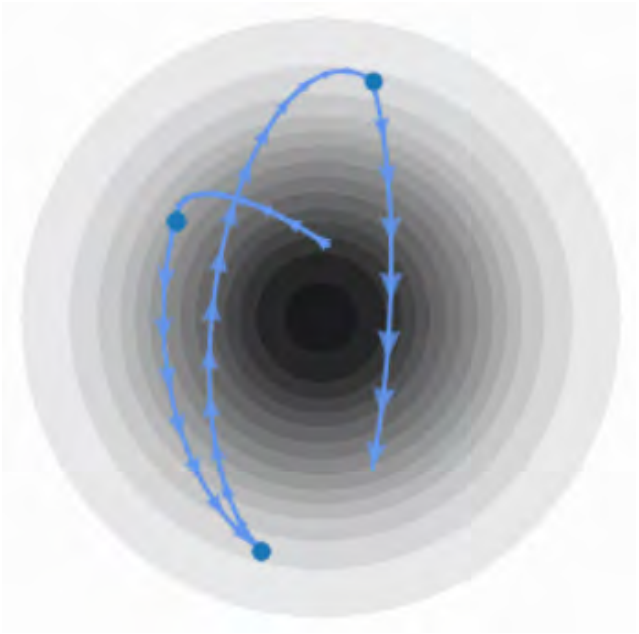
Adrian Lundell



Iakov Kharitonov

Probability sampling inspired by fields

Morten Holm, Tiziano Barbari



Stochastic parameter fitting is resource-consuming: Stan already offers great algorithms for Bayesian inference, but let us try to improve them anyway, with code optimisation and parallelism!

The Stan core library offers several high-performance inference algorithms written in C++, used for scientific modelling. We tried to enhance the performance of the Hamiltonian Monte Carlo (HMC) algorithm, enabling the algorithm to run on distributed systems using MPI as well as shared-memory multiprocessing programming via OpenMP.

Introduction

The purpose of our project was the implementation of a more efficient, parallel, and scalable, Hamiltonian Monte Carlo; we started with an existing statistical framework and an already implemented C++ code skeleton, and tried to improve the data structure and the integration methods. Simulating an ensemble using a Structure of Arrays (SoA) approach will also enable GPGPU processing more readily down the line. Let us briefly illustrate with a basic example what **stochastic parameter fitting** is, and how it relates to Stan; the "*coin toss*" example that we used for testing consists in:

- data (the outcome of 2 coin toss experiments):
  y1 = (0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1)
  (balanced coin)
  y2 = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
  (totally unbalanced coin).
- the model and the wanted properties of the parameters to be computed:
  $\theta_1,\ \theta_2 \sim beta(1,1)$
  $\theta_1,\ \theta_2 \in [0,1]$
  $y_1 \sim Bernoulli(\theta_1)$
  $y_2 \sim Bernoulli(\theta_2)$.

Such a declared model is first read by Stan, then translated into C++ code, and finally used in conjunction with any C++ algorithm that we devise and feed to Stan itself, specifying its intended use. The algorithm we built is an HMC method able to sample from a posterior distribution (Bernoulli, in our example); the framework uses our code to return the final parameter estimations, which for our toy model should be approximately $\theta_1 \sim 0.5$ and $\theta_2 \sim 1$.

Theory

MCMC. Now let us give a bit of context to our HMC implementation: HMC is a very special Markov Chain Monte Carlo (MCMC) algorithm. MCMC methods sample from a continuous random variable (RV) and we can use these samples to compute integrals over that RV, such as its expectation or its variance. How, you ask? Sampling means that a point starts from an arbitrary position and is evolved by sequentially applying the so-called *Markov transitions*; what we obtain are random "walkers" which move around and try to reach points that can contribute significantly to the integral computation. Now, it turns out that, in high dimensions, the density and the volume (that make up the expectation) undergo a tension that forces the highest probability into a narrow region of space, called *typical set*. Unfortunately, such a *walker* will have a hard time approaching the typical set, and often exhibit random walk behaviour (and hence a slow convergence) and autocorrelation.

HMC. Here is the focus of our project, where HMC comes into play! This algorithm introduces a fictitious physical system, equivalent to the statistical one, and simulates Hamiltonian dynamics in it[1]! The samples are treated as physical particles, having a *position* **q** (a vector concatenating the statistical parameters) and a (randomly initialised) *momentum* **p**. The particles (or samples) obey Hamilton equations and, ideally, follow the Hamiltonian flux, quickly reaching the typical set and efficiently exploring it. Numerically, we have to integrate, of course! Given that we are in a Hamiltonian context, we do it with *symplectic integrators*, which preserve phase-space volume; although energy is not exactly conserved, energy fluctuations are bounded:[3] hence, the computed orbit stays close to the true orbit even for long integration times. A general formula for symplectic integrators can be obtain starting from the Hamiltonian, assuming that it is separable:

$$H(p,q) = T(p) + V(q) \qquad (1)$$

A simplified form of Hamilton equations are the building blocks of an order k symplectic integrator, which executes the following update steps k times:

$$q_{i+1} = q_i + c_i \frac{p_{i+1}}{m} t \qquad (2)$$

$$p_{i+1} = q_i + d_i F(q_i) t. \qquad (3)$$

where c and d depend on k and i, and $F(q)$ is proportional to $V(q)$, which is, equivalently:[2]

- the positional negative potential energy function

or

- the logarithm of the joint density of the components of q.

Important examples are:

- the 1^{st} order *symplectic Euler* ($c_1 = d_1 = 1$), not time-reversible;
- the 2^{nd} order *Störmer–Verlet* ($c_1 = 0, c_2 = 1, d_1 = d_2 = 1/2$), symmetric in time and of which the leapfrog is a clever variant.
- the 3^{rd} order *Ruth*: ($c_1 = 1$, $c_2 = -2/3$,$c_3 = -2/3$, $d_1 = -1/24$, $d_2 = 3/4$, $d_3 = 7/24$). This, and another 4^{th} order integrator by Ruth were initially developed for, and distributed among, the particle-accelerator community.

Profiling. Extrae is a package developed by BSC (Barcelona Supercomputing Center) and is used for tracing programs compiled and run with OpenMp, MPI, or both. Specifically, it "generates trace files that can be later visualized with Paraver" (from tools.bsc.es), a powerful performance visualisation and analysis tool. How? It inserts probes in the application that we want to analyse, in order to collect information useful for determining the performance.
Paraver has two main ways of displaying its performance analysis, that pertain to two different types of information:

1. the *timeline display* shows how the code behaves in time and with respect to processes, making it easy to detect patterns;
2. the *statistics display* analyses the "data that can be applied to any user selected region", indicating precisely what can be optimised, and how.

Interestingly, Paraver can display several types of objects, and they relate to the parallel programming model concepts (workload, application, task and thread) and to resources group (system, node and CPU).
Parallelisation. We employed MPI that, for each node, would generate its own ensemble and we performed a reduction step according to a summation operation using `MPI_SUM`

```
MPI_Reduce(send_data, recv_data,
                ..., MPI_Op op, ... );
```

In the end we had to have the 0 rank write all the data into a final matrix of particle positions. Furthermore, OpenMP was utilized, using common *pragma directives*. Since this is an integral part of the project, let's illustrate how exactly this works. Using the simplest case, we have a for loop; when it is time for parameter estimations, we create a matrix with the computed particle positions. To do that, we call:

```
# pragma omp parallel for
for (Id=0; Id<localNumParticles; ++Id) {
```

Utilising `#pragma omp` enables the compiler to handle the parallelisation and, further, we can enable parallelism using the `parallel` construct. We can use the `for` clause to indicate a for loop that will tell the compiler to split up the job to several threads. The function that initialises the ensemble of particles (using a Gaussian distribution for the positions and a thermal distribution for the momenta) is created in a similar manner as follows

```
#pragma omp parallel for collapse(2)
for (Id=0; Id<numParticle; ++Id){
    for (param=0; param<numParam; ++param)
```

Here the clause `collapse(2)` indicates a nested for loop and will cause the nested for loop to be combined into a single for loop, meaning that each index operation is independent of each other. S.t. `for i=1:N; for j=1:M` becomes `for k=1:N*M`. This structure has the advantage of a parallel environment without write conflicts: each thread will have its own objects!

Methods

Our development is a subbranch of the Stan Open-source software via a private Github Repository; the programming, testing and profiling were done on a Cluster belonging to the Institute for Theoretical Physics at the Hartree center.
How to work with Stan. Stan can interface with several programming languages; we used CmdStan, the shell interface. Through the shell, we ran the tests, performed the stochastic fitting, accessed supercomputing resources, and profiled. Therefore, we had a standard pipeline of commands that we routinely executed:

1. load needed modules: `binutils 2.30`, `cmake 3.20.0`, `boost 1.72_gcc7`, `eigen 3.3.9`, `mpich 3.2.1`;
2. get updates from the repository;
3. build the code, test, profile;
4. update the repository.

The first build in step 3 is for using stan itself and is here compiled with 5 cores:

```
$ make build -j 5
```

A few moments later, Stan is ready to build a solution (our example), so let us do it!

```
$ make examples/coin_toss/coin_toss
```

For explaining the process, the *stanc* compiler reads the *.stan* model and produces a C++ class (*.hpp*).
Finally, "the C++ compiler compiles all C++ sources and links them together with the CmdStan interface program and the Stan and math libraries" (to quote the CmdStan User's Guide), and creates an executable *coin_toss.o* which will produce the wanted estimations.

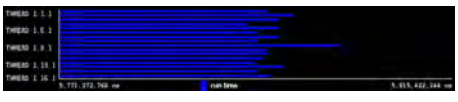


Figure 1: Profiling using Extrae; Displayed (blue lines) are the runtimes for each of the 16 thread in a selected area of a process.

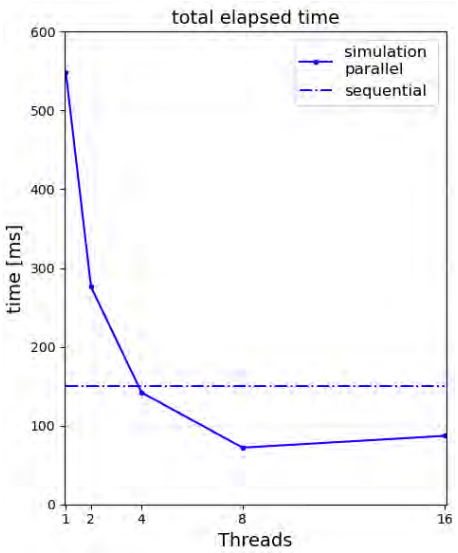


Figure 2: Displayed are the runtimes for the parallelised simulation (solid line) and the sequential time without parallelisation enabled (dashed line).

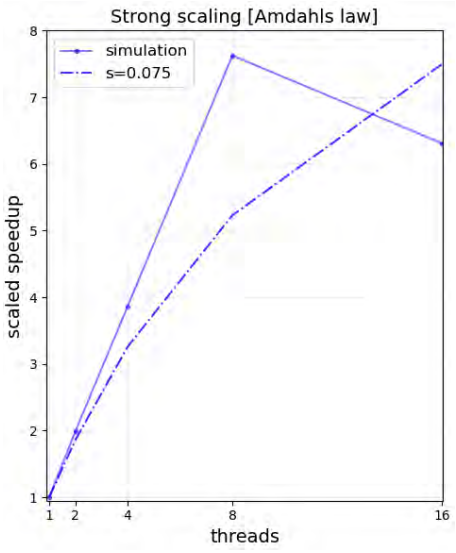


Figure 3: Strong scaling using 16384 particles for increasing thread count with s being the serial part of the code according to Amdahl's law.

Results

Actual coding. Stan provides algorithms for estimating statistical parameters: our job, in a nutshell, consisted in focusing on a single part of the process, namely the efficient sampling from posterior distributions. We tried and devised a more efficient HMC, starting from a sophisticated C++ code skeleton. Our job consisted in completing some parts of the skeleton and implementing new ones; in particular we created or completed:

- the integrator structure, in particular the integrating step, mainly using a leapfrog scheme and later also 2^{nd} order Störmer-Verlet. Thus allowing us to run the simulation for different integration schemes;
- an ensemble of particles, following the Structure-of-Array model (a single structure with one array per particle feature);
- the HMC sampling algorithm;
- the estimation of the parameters of the model;
- Parallelisation via MPI and OpenMP.

The project involved parallelisation, code optimisation and performance engineering, and we achieved this by means of code restructuring and profiling tools (Extrae/Paraver). Our implementation of HMC turned out to be satisfactory, fast and producing meaningful results for a large linear normal model, a result not obtained by the previous implementation!
An initial simulation with 2^{14} particles, evenly spread out on threads$=[1, 2, 4, 8, 16]$, showed promising performance gains, using multiple threads.

Discussion

An issue with OpenMP and the integrator was encountered and unresolved, most probably a framework issue with Stan; it would require some investigation, because resolving this issue would increase performance. For the working OpenMP, as seen on figure 3, a larger number of threads results in a performance boost, and a greater boost can occur testing for increased node count. Due to a maintenance of the HPC systems we could not retrieve the data to show here, though. In profiling, as figure 1 shows, we can find sequences where the code stalls, i.e. several threads wait for others to finish before all threads collectively move on. Smaller batches of sequences can be used, not to avoid but to minimise the possibility that a sequence becomes a bottleneck for various reasons. In figure 2 we see a significant overhead from the difference in run time (simulating with a single thread). This is not really an issue, as even most basic computers now have at least 4 threads, which is where the break-even point is seen. Not shown here, we observed that 1024 particles per thread is the sweet spot when performing the simulation: the scaled speedup would peak at 4 threads using 4096 particles and at 2 threads using 2048 particles. This does not reflect, though, the outcome of figure 3, where we run the simulation for 16384 particle and it should be seen that the peak is at 16 threads, whereas the cause for this is currently unknown.

Conclusion

Our implementation shows promise for applications in Bayesian inference and is more performing than the former one, and could be scalable: this may be the base for future work with further parallelisation and extensions of the algorithm.

Acknowledgements

An exabyte of thanks to our mentor, Anton, for his patient support and skills that he let us develop: software development, statistics, data analysis, and profiling; to PRACE for the opportunity to work with HPC resources; to our loved ones and whoever contributed to make this experience possible.

References

[1] Michael Betancourt (2018). A Conceptual Introduction to Hamiltonian Monte Carlo

[2] Matthew D. Hoffman and Andrew Gelman (2014) The No-U-Turn Sampler: Adaptively Setting Path Lengths In Hamiltonian Monte Carlo

[3] http://www.cs.toronto.edu/~wayne/research/thesis/depth/node7.html

PRACE SoHPCProject Title

Scaling HMC on large multi-CPU and/or multi-GPGPUs architectures

PRACE SoHPCSite

Hartree centre, UK

PRACE SoHPCAuthors

Morten Holm, Niels Bohr Institute, Denmark
Tiziano Barbari, Padua University, Italy


Morten Holm


Tiziano Barbari

PRACE SoHPCMentor

Anton, Lebedev, Hartree Centre, UK
Vassil, Alexandrov, Hartree Centre, UK

PRACE SoHPCContact

Morten Holm, Niels bohr institute
Phone: +45 2714 4489
E-mail: qgf305@alumni.ku.dk
Tiziano Barbari, Padua University
Phone: +39 320 2391936
E-mail: tizianobarbari@gmail.com

PRACE SoHPCSoftware applied

MobaXTerm, Github, Visual Studio 2019 (C++), Extrae/Paraver

PRACE SoHPCMore Information

www.virtouso.org

PRACE SoHPCAcknowledgement

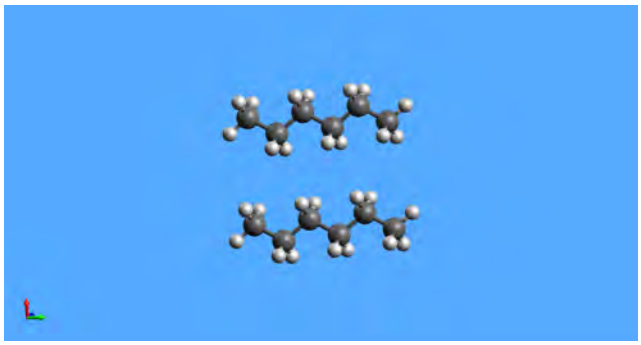
Thanks to the team at Hartree and especially Anton and Vassil

PRACE SoHPCProject ID

2125

Scientific Benchmark

Sahin Can Alpaslan



I felt like I was entering an incredible universe at a hpc workshop a few months ago. Since then I have been doing my best to explore this universe.

Abstract

The goal of my project is to get acquainted with different relevant software for scientific simulations and be able to analyse and describe the performance of the most relevant applications that are currently run by researchers in the field. The performance results will give out a comprehensive summary of the best compilation options that can be used to achieve maximum performance for different types of simulations on CPUs and GPUs.

```
TOTAL RUN TIME: 0 days 0 hours 31 minutes 0 seconds 739 msec

JOB STATISTICS
==============
Job ID: 10715074
Cluster: sara
User/Group: sahina/sahina
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 24
CPU Utilized: 11:52:05
CPU Efficiency: 95.560f 12:25:12 core-walltime
Job Wall-clock time: 00:31:03
Memory Utilized: 44.51 GB
Memory Efficiency: 73.810f 60.31 GB
```

Figure 1: 1 Node Calculation

Introduction

The data from this Benchmark will be used to understand better how to use the resources of our HPC center. Will allow us to guide better the users in how to run their simulations. Comparison between different packages will allow us to guide the users for the best code to use depending on the system size. Produce template scripts to be used CC calculations on large systems. These results will be used to model the reaction mechanism of HIV1 protease with a larger QM layer. Ab initio quantum mechanical methods(QM) are now able to provide reasonably good values for chemical optimization. We prefer this method because the QM results give high accuracy.

Why we need HPC?

Short answer is we have a limited lifespan. The long answer is for many purposes such as computational chemistry the ability of process data in real time is crucial. Although the performance of personal computers is on the rise, it may not be sufficient for calculations. We can devote the time we gain to more analysis or problem solving. We can enlarge the molecular systems we work with or obtain results closer to reality. That's why we use hpc systems.

Methods

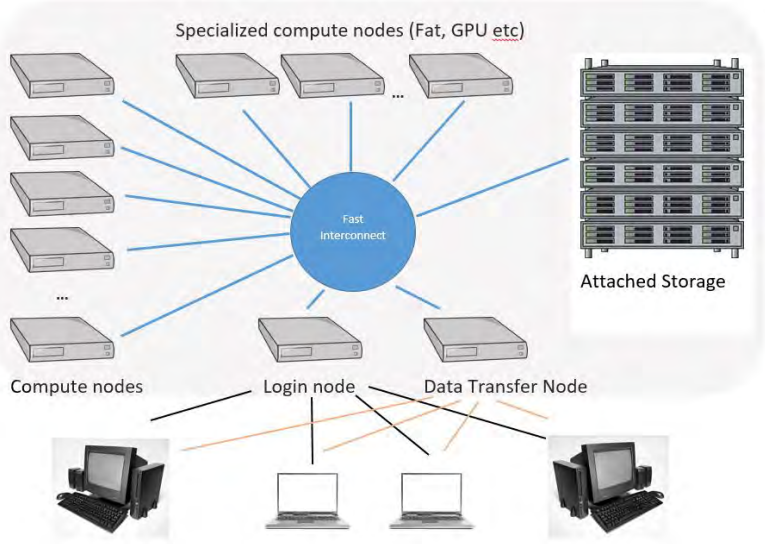
Our molecule system contain 40 atoms and using correlation consistent triple zeta (cc-PVTZ) basis set and dlpno ccsd(t) method. The 'cc-p', stands for 'correlation-consistent polarized' and the 'V' indicates they are valence-only basis sets. They include successively larger shells of polarization (correlating) functions (d, f, g, etc.) 'T Z' triple-zeta. This basis functions use for first and second row atoms.

```
TOTAL RUN TIME: 0 days 0 hours 21 minutes 29 seconds 301 msec

JOB STATISTICS
==============
Job ID: 10715240
Cluster: sara
User/Group: sahina/sahina
State: COMPLETED (exit code 0)
Nodes: 2
Cores per node: 24
CPU Utilized: 16:47:13
CPU Efficiency: 97.520f 17:12:48 core-walltime
Job Wall-clock time: 00:21:31
Memory Utilized: 43.10 GB
Memory Efficiency: 35.730f 120.62 GB
```

Figure 2: 2 Node Calculation

Dlpno ccsd(t) method is domain-based local pair natural orbital (DLPNO) CC methods that allows coupled cluster calculations to be performed on larger systems than ever before. We use 2 packages for benchmarking: ORCA[1] and Psi4[2]. Orca enables to run ab initio molecular dynamics (AIMD) simulations of small to medium-sized (non-periodic) systems, using all the different electron structure methods. Psi4 is an open-source suite of ab initio quantum chemistry programs designed for efficient, high-accuracy simulations of molecular properties. The program can compute energies, optimize molecular geometries, and compute vibrational frequencies.



Discussion & Conclusion

Benchmarking is the practice of comparing processes and performance metrics and a way of discovering what is the best performance being achieved. For this reason, we performed benchmark tests using different number of nodes. When using HPC clusters, it is almost always worthwhile to measure the scaling of your jobs.Our molecule system contain 40 atoms, 100 electrons and 752 atomic orbital. So we can say that we are working on a large system. Scalability is defined as the ability to handle more work as the size of the computer or application grows. For HPC clusters, it is important that they are scalable, in other words the capacity of the whole system can be proportionally increased by adding more hardware. As you can see on the Table 1, The system that we are working on can scale. But it shows strong scaling feature. Strong scaling, the number of processors is increased while the problem size remains constant.Increasing the number of nodes after a point has no effect on execution time. Ok but what causes this? For real speedup the speedup after 4 nodes gets worse. Because of increasing data traffic between the compute nodes, adding another node can be a disadvantage and will even slow down the application. It can also be so said that the compute time also cannot be reduced further as the problem size cannot be further divided to improve it for a effective reduction in time for data traffic and I/O. So our system showed low strong scaling feature. Using low and strong together can be confusing, don't be confused by this. It is strong because our processing time decreases when we increase the number of nodes, but it is also low because it starts to remain stable after 4 nodes.

Table 1: Execution Time

Node	Execution Time(min)
1	31
2	21
3	18,30
4	18,50
5	17,39
6	18,25
7	19,14

Future Work

I feel really lucky to be involved in this project. For me, simulations and coding mean constant problem solving. HPC systems also save us time in these solutions. In this way, we can spend more time solving the problem instead of waiting. As I mentioned above, we could not achieve a linear scaling in the project. After 4 nodes the execution time remained the same value. I'll dive deeper and investigate why. I will try to find answers to questions such as: Why are some systems scaling while others are not?, can we specify a general method? How do we use automation in calculations? Experting the hpc architecture and the mpi method seems like a good start.

Acknowledgements

I am deeply thankful to my project mentor Ana Maria da Cunha for all support and interest through the two months and my teammate Milana Mirkovic. PRACE for giving us a place on the Summer of HPC programme. Special thanks to Leon Kos, Pavel Tomsic for all their work and SurfSara B.V for allowed to use Cartesius System.

References

[1] Neese, F. (2012) The ORCA program system, Wiley Interdiscip. Rev.: Comput. Mol. Sci., 2, 73-78. Neese, F. (2017) Software update: the ORCA program system, version 4.0, Wiley Interdiscip. Rev.: Comput. Mol. Sci., 8, e1327.

[2] "Psi4 1.4: Open-Source Software for High-Throughput Quantum Chemistry", D. G. A. Smith, L. A. Burns, A. C. Simmonett, R. M. Parrish, M. C. Schieber, R. Galvelis, P. Kraus, H. Kruse, R. Di Remigio, A. Alenaizan, A. M. James, S. Lehtola, J. P. Misiewicz, M. Scheurer, R. A. Shaw, J. B. Schriber, Y. Xie, Z. L. Glick, D. A. Sirianni, J. S. O'Brien, J. M. Waldrop, A. Kumar, E. G. Hohenstein, B. P. Pritchard, B. R. Brooks, H. F. Schaefer III, A. Yu. Sokolov, K. Patkowski, A. E. DePrince III, U. Bozkaya, R. A. King, F. A. Evangelista, J. M. Turney, T. D. Crawford, C. D. Sherrill, J. Chem. Phys. (2020). (doi: 10.1063/5.0006002).

Figure [3] https://www.hpc.iastate.edu/guides/introduction-to-hpc-clusters/what-is-an-hpc-cluster


Sahin Can Alpaslan

PRACE SoHPCProject Title
Benchmarking Scientific Software for Computational Chemistry in The Dutch National Supercomputer

PRACE SoHPCSite
Republic of Turkey

PRACE SoHPCAuthors
Sahin Can Alpaslan, Republic of Turkey

PRACE SoHPCMentor
Ana Maria de Carvalho Vicente da Cunha, Surf Sara, Netherlands

PRACE SoHPCContact
Sahin Can Alpaslan
Phone: +90 5374976194
E-mail: sahincanalpaslan@gmail.com

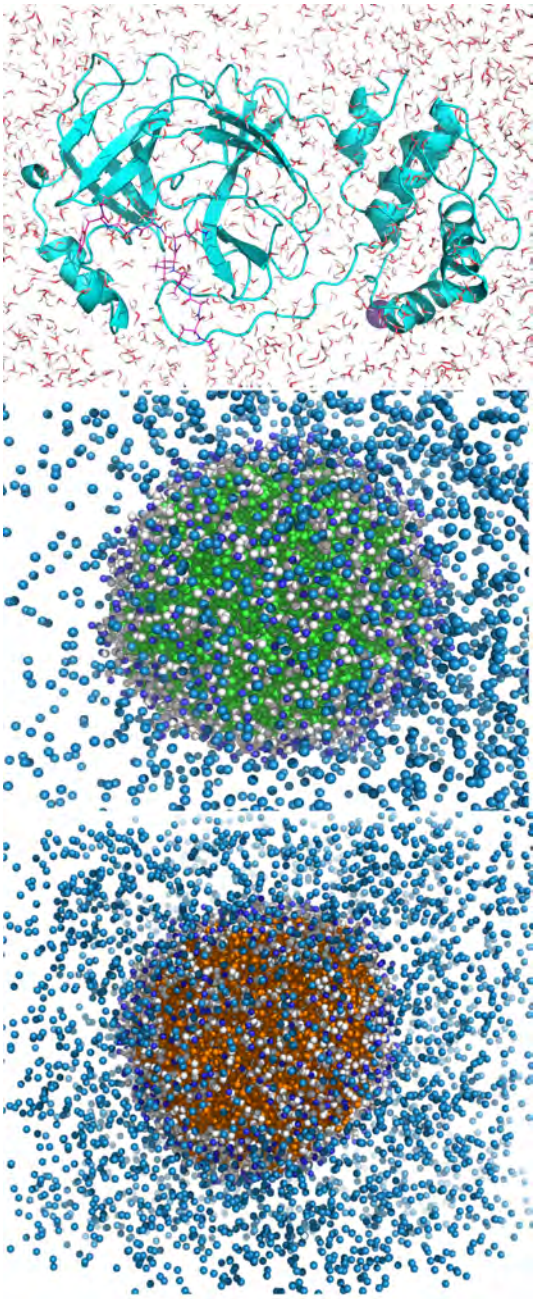
PRACE SoHPCSoftware applied
ORCA, PSI4

PRACE SoHPCMore Information
ORCA PSI4

PRACE SoHPCProject ID
2126

Benchmarking Molecular Dynamics Simulations

Milana Mirkovic



Molecular dynamics simulations are a powerful tool for studying the dynamics of biomolecular systems. Thanks to high-performance computing, we can now easily simulate large systems and longer time scales. However, code parallelization introduces a new layer of intricacy to the setting up of a simulation. Benchmarking is a great way to get a clear overview of the best setups and so help the users achieve maximum performance.

The conventional experimental methods, while important, cannot provide insight into processes happening at the atomic level and on submicrosecond time scales. Molecular dynamics (MD) simulations act as a "computational microscope" that allows us to overcome these limitations.[1] To simulate a biomolecular system, we need a model and a force field. A model describes a molecule as a collection of points that are assigned coordinates in Cartesian space, mass, and charge. The point can be equivalent to an atom in the case of an all-atom (AA) model[2] (Figure 1A), or it can represent a group of atoms, acting as a "pseudo-atom", in the case of a coarse-grained (CG) model[3] (Figure 1B). How these points interact is described by a force field,[4] which consists of a set of equations, called potential energy functions, and parameters that are used in them. The computer solves these equations for every point, calculates the forces acting on them, and updates the coordinates. The type of model dictates what kind of a force field we are using, which in turn dictates the set of parameters. It also greatly influences the performance — it takes a lot of computational resources to simulate an all-atom system. By simplifying the representation, we also reduce the number of degrees of freedom, allowing us to simulate bigger systems and longer processes.

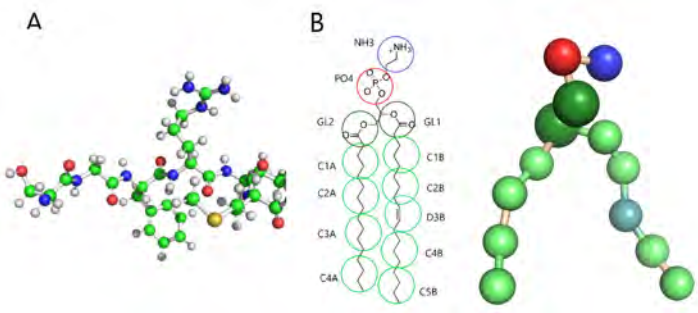


Figure 1: A. An all-atom model B. A coarse-grained model. Left is the skeletal formula of the coarse-grained model on the right. Superimposed are the correspondent pseudo-atoms.

One of the most popular simulation software packages is GROMACS,[5] which comes with plenty of options to optimize simulation times, especially on highly parallel systems. Parallelization is implemented the domain decomposition algorithm[6] (Figure 2). Since running a simulation on a single core would take too long, we can parallelize it by decomposing the simulation box into domains. Each domain is then assigned

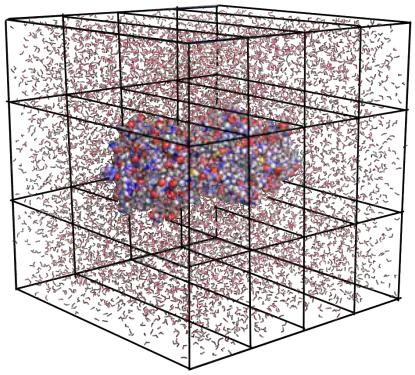


Figure 2: A domain decomposition grid of 4×3×1 cells on 12 MPI ranks.

to an MPI rank, which calculates the interactions within the domain. Some of the calculations can also be offloaded to a GPU, which can further improve the performance. Additionally, the way GROMACS was compiled can also influence the performance. However, the scaling is not always linear and depends on the size of the system and the type of algorithm used. Since most of today's MD simulations are done on HPC systems, it is therefore absolutely necessary to perform extensive benchmarks of representative all-atom and coarse-grained systems, as this data helps guide the HPC users how to best use the available resources.

Methods

The simulations were run on the supercomputer Cartesius at SURFsara. The benchmarked nodes were the CPU nodes of type "Haswell" and "Broadwell", with 24 and 32 cores per node respectively, and the GPU nodes, with 16 cores per node and 2 NVIDIA Tesla K40m GPUs. We used GROMACS 2020.3 simulation package, compiled with either the foss or the intel toolchain. The simulated systems were as follows: an all-atom representation of the main protease of SARS-CoV-2 (COVID protease) bound to a peptide,

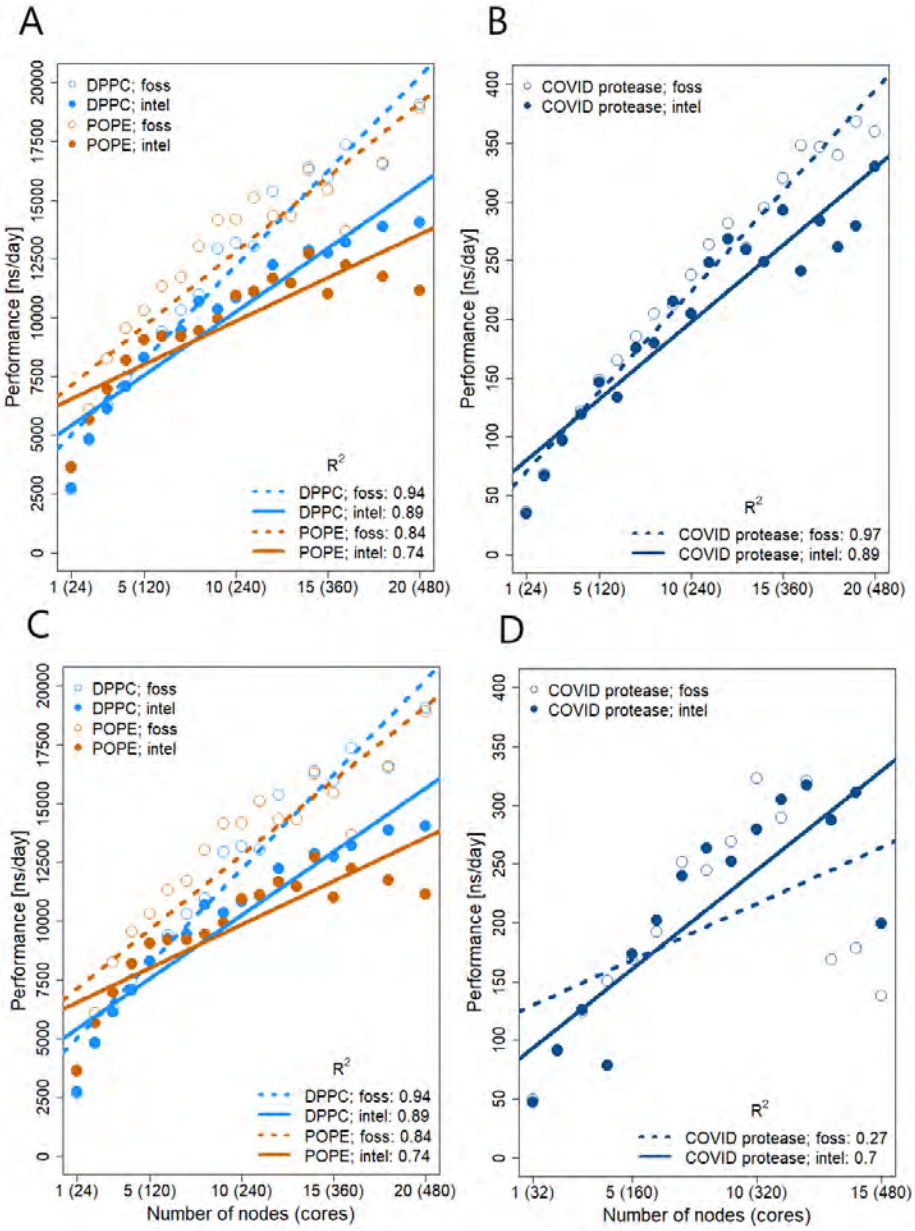


Figure 3: Comparison of foss and intel toolchains. A. Coarse-grained algorithm and B. all-atom algorithm on "Haswell" nodes. C. Coarse-grained algorithm and D. all-atom algorithm on "Broadwell" nodes.

with 98632 atoms, and coarse-grained representations of DPPC and POPE vesicles, with 72076 and 72953 pseudo-atoms respectively. These systems were chosen as their sizes are representative of the systems an average HPC user simulates. The performance was measured in nanoseconds per day (ns/day) we could simulate. The scalability was assessed by calculating R^2.

Results & Discussion

We compared scalability of all-atom and coarse-grained algorithms on CPU and GPU nodes. Additionally, we benchmarked two different versions of GROMACS, built with either the foss or the intel toolchain.

In general, all three systems show good scaling on all benchmarked node types. When comparing CPU nodes, the nodes of type "Haswell" seem to offer more reliable performance, as shown in Figure 3. However, as the number of cores increases, the scaling becomes suboptimal. The cutoff point seems to be >300 cores across all systems. The foss toolchain performs better in terms of both the ns/day achieved and the scalability. It only seems to perform worse for the all-atom algorithm on "Broadwell" nodes (Figure 3D). There is a significant drop in performance if the simulation is run on more than 12 nodes, for example, it drops from ~320 ns/day for 12 nodes to ~140 ns/day for 15 nodes. The intel toolchain seems to perform slightly better on the GPU

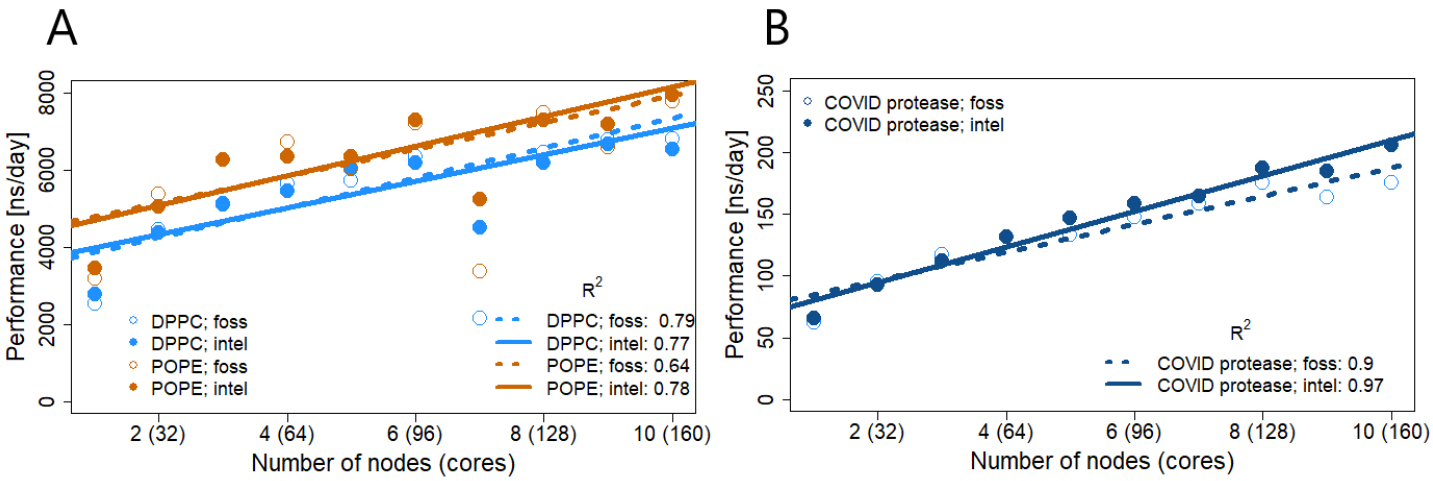


Figure 4: Comparison of foss and intel toolchains on GPU nodes. A. Coarse-grained algorithm and B. all-atom algorithm.

nodes, as shown in Figure 4. The drop in the performance on 7 nodes for the coarse-grain algorithm is due to sub-optimal domain decomposition in the coarse-grained systems for this node configuration. The overall better performance of the foss toolchain could be partly explained by using different libraries for fast Fourier transform. The foss toolchain comes with FFTW, which specifically for GROMACS is often faster than Intel MKL.

In general, the all-atom algorithm scales better than the coarse-grained algorithm but does not achieve as good of a performance, which is expected. There is some difference in the performance between the two coarse-grained systems, which can be explained by the difference in the number of atoms and the way they are distributed in the simulation box.

There is also variation in the performance depending on how the simulation box was decomposed and how many OpenMP threads were assigned per MPI rank. As we can see in Figure 5, the best performing setup for the all-atom algorithm on "Haswell" nodes and GROMACS compiled with the foss toolchain is 1 OpenMP thread per MPI rank in most cases. However, it is often that 2 OpenMP threads per MPI rank perform just as well or better. The variation in performance increases with the increasing number of nodes, with the difference between the best and worst performing option reaching almost 180 ns/day on 16 nodes.

References

[1] Dror RO, Dirks RM, Grossman JP, Xu H, Shaw DE. Biomolecular simulation: a computational microscope for molecular biology. Annu Rev Biophys. 2012;41:429-52. doi: 10.1146/annurev-biophys-042910-155245.

[2] http://cmt.dur.ac.uk/sjc/thesis_dlc/node45.html

[3] Kmiecik S, Gront D, Kolinski M, Wieteska L, Dawid AE, Kolinski A. Coarse-Grained Protein Models and Their Applications. Chem Rev. 2016;116(14):7898-7936. doi: 10.1021/acs.chemrev.6b00163.

[4] https://manual.gromacs.org/current/reference-manual/functions/force-field.html

[5] https://www.gromacs.org/

[6] Hess B, Kutzner C, van der Spoel D, Lindahl E. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. J. Chem. Theory Comput. 2008;4(3):435–447. doi: 10.1021/ct700301q.

[7] Ullrich S, Nitsche C. The SARS-CoV-2 main protease as drug target. Bioorg Med Chem Lett. 2020;30(17):127377. doi: 10.1016/j.bmcl.2020.127377.

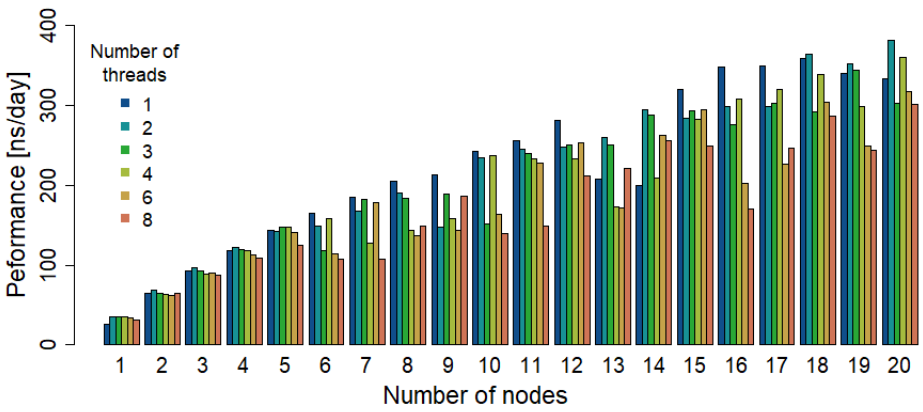


Figure 5: Performance of all-atom algorithm on "Haswell" nodes depending on the number of OpenMP threads per MPI used. Toolchain: foss.

PRACE SoHPCProject Title

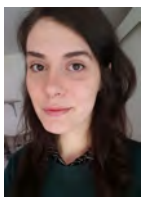
Benchmarking Scientific Software for Computational Chemistry in the Dutch National Supercomputer

PRACE SoHPCSite

SURF, Netherlands

PRACE SoHPCAuthors

Milana Mirkovic, University of Vienna, Austria


Milana Mirkovic

PRACE SoHPCMentor

Ana Maria de Carvalho Vicente da Cunha, Vrije Universiteit Brussel, Belgium

PRACE SoHPCContact

Milana Mirkovic, University of Vienna, Austria
E-mail: mirkovic.milana@outlook.com

PRACE SoHPCSoftware applied

GROMACS

PRACE SoHPCMore Information

www.gromacs.org/

PRACE SoHPCAcknowledgement

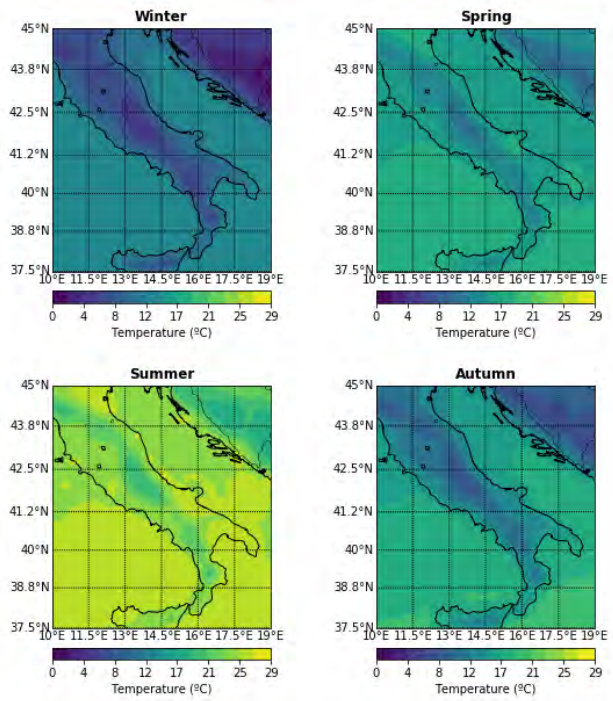
Many thanks to my mentor Ana Maria de Carvalho Vicente da Cunha for her time and valuable input, my teammate Sahin Can Alpaslan, SURFsara for providing the resources for the project and PRACE for organisation.

PRACE SoHPCProject ID

2126

Data Storage in Cloud computing

Maria Li López Bautista
João Quintiliano Sério Guerreiro



The innovation of cloud has become a convenient method to store data online and, faced with many storage services providers, there is interest in maximizing its data processing efficiency.

Nowadays, the use of the cloud to conduct scientific research when working with large amounts of data is becoming the norm since it provides an easy way to adapt the system resources to our needs. So, therefore serving as a cost-saving and a sustainable approach.

However, data handling can be a very demanding task given that is dependent not only on the system resources, but also on the type and size of data that is being dealt with. Here is when **Cloud storage** comes in handy because it allows researchers to store all the large information that want to work with without compromising any disk storage. Thus, it is of great importance to study how distinct storage types cope with assorted datasets and how different implementations can lead to substantial **performance boosts** of the data processing tasks. In our case, performance is how fast executes a problem with the data stored in a determinate storage service.

With this idea in mind, the main goal of the project is to study and implement real life cases in different services and applications, and compile advice of the best practices of the different storage systems so that, ultimately, the best option can be picked. In other words, the major objective is the creation of a catalogue with a collection of reference implementations of pipelines for real life cases, using different storage services.

SURF Research Cloud

In order to set up the programming environment we used SURF Research Cloud,[1] which is a collaboration portal for research used to build virtual research workspaces. A workspace is a research environment where a user can select an application along with the data that want to work on, and with that deploy a **virtual lab** in the cloud for the user and their colleagues to work in. In fact, one can use preconfigured workspaces and datasets or add its own. Furthermore, institutions, research communities and service providers can contribute to the service functionality by integrating compute and storage resources.

Other advantages of using SURF Research cloud is that we can:

- Create reproducible research environments.
- Use the best resources for our research.
- Work together securely.

In our case, we deployed JupyterLab virtual machines of different flavours (different number of cores and memory capacity) since we were working with Python.

Storage services

For the purpose of comparing and contrasting the impact of different storage systems on performance, also known as **benchmarking**, the SURF organisation provided us with several storage services in addition to the local memory of the virtual machine. So far, we have attempted to work with the following ones:

SURF Research Drive

SURF Research Drive[2] is a cloud-based shared environment that is specifically designed for requirements like large storage quotas and collaborative work with educational and governmental institutions or external private parties.

In summary, it provides a straightforward way to securely and easily store and share research data.

dCache

dCache[3] is a distributed storage for scientific data. Roughly speaking, it is capable of managing the storage and exchange of several hundreds of terabytes of data, transparently distributed among dozens of disk storage nodes.

Methods

Given the objective in assessing how different storage types affect the performance of data processing first, it was necessary to have some data and to know how to handle it in a Jupyter Notebook environment in Research Cloud. After that, some examples on how to process the data were sketched constituting the use cases to be tested. After having implemented the solutions for all the sketched sample problems, these were tested on the same data that was stored in the local storage of the created VM's (Virtual Machines) in Research Cloud, Research Drive, dCache and iRODs. The latter three storage systems were made connected to our workspaces by webDAV communication protocol thus, the information could be easily accessed. To be able to compare all of these, the solutions were executed several times for each storage system, and the execution times were registered. Lastly, a benchmark analysis provided insight over the efficiency of each storage.

Dataset

In this project we have worked with climate data from the ERA5 dataset, provided by ECMWF (European Centre for Medium-Range Weather Forecasts).[5] ERA5 provides hourly estimates of a large number of atmospheric, land and oceanic climate variables (from 1979 to present). The data covers the Earth on a 30 km grid and resolves the atmosphere using 137 levels from the surface up to a height of 80 km. ERA5 also provides information about uncertainties for all variables at reduced spatial and temporal resolutions.

This data is downloaded either from the Climate Data Store (CDS) website or from API requests, the latter having been used for downloading directly to the storage service in the cloud. It allows for specifying a desired time range, region and spatial resolution and the data is stored in NetCDF files with hourly values for all the desired weather variables that one wants to work with. In this project we have selected the 2 meter surface temperature to be our working variable. As for the working region, we have downloaded data correspondent to central Europe but have only chosen the region of Italy to be analyzed. With regards to time range, data from 1979 to 2020 was considered.

Use cases - Sample problems

Typically there are two modes of doing data processing which are sequential reading of data and targeted access of data. The **sequential reading of data** consists in accessing all of the data's information, sequentially, and filtering it if it is wanted for processing or not. The **targeted access of data** requires some pre-processing which is responsible for determining the location of the information of interest in the data. Afterwards, the accessing and processing of information is only made for those specific locations in the data. In order to approach both of these modes, three separate problems were considered:

- **Problem 1** - Calculating the average of a parameter for the whole history (sequential reading).
- **Problem 2** - Calculating the average of a parameter for a specific moment in time (targeted access).
- **Problem 3** - Calculating the average of a parameter per each of the 4 seasons (sequential reading and/or targeted access).

For the calculation of the average three different approaches were implemented to see which of these would result in the most efficient method for the different storage types and problems considered. One of the methods includes resorting to an **Online Algorithm** - one that can process its input piece-by-piece in a serial fashion, i.e, in the order that the input is fed to the algorithm. In addition, python's `multiprocessing` package was explored, bringing parallelization into the picture which allowed us to work with multiple data files simultaneously.

Benchmark Analysis

Benchmarking is the process of determining the best processes, strategies and techniques to achieve the business goals. In this project's case, it is to choose the best compute and storage infrastructures for the research. For that, we've tested the storage systems by running several times all of the implemented approaches for each problem over the same data stored in the different storage services explained before. Then, the respective mean and standard deviation of the running times are calculated and plotted, allowing a comparative analysis. Furthermore, when working with multiple data files simultaneously (introducing parallelization), the number of processes was changed and its running times analyzed.

Preliminary results

The end product of the benchmark analysis is that all three algorithms based on the calculation of the average per cell work faster when data is stored locally than in dCache or Research Drive, as shown in Figure 2. Likewise, it has to be noted that all the results showed are from Problem 1 since the behaviour of time with total number of processes was always the same regardless the problem treated.

On the other hand, in the following figure it is noticeable how the increase of the number of processes leads to a decrease in the run time of all three algorithms.

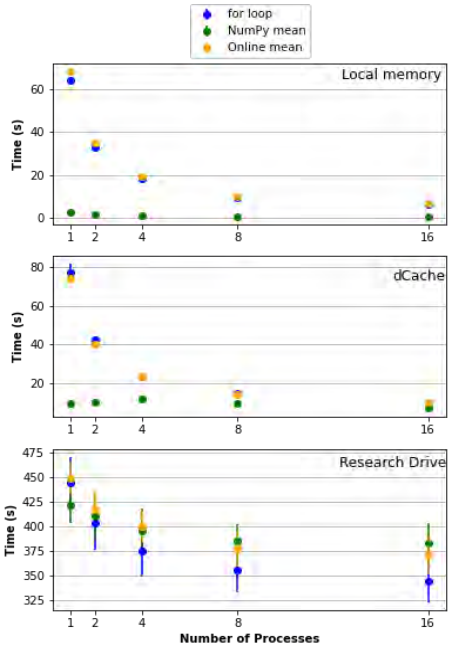


Figure 1: Graphical representations of the resulting mean time values with their corresponding errors for each storage system.

Notwithstanding that, we can also note that the decrease of time when implementing the "for loop" and "Online mean" algorithms is deeper than when using the "NumPy mean" method. The reason because the parallelization/multiprocessing approach (when working with several files simultaneously) has a major effect in the first ones is probably due to a RAM issue.

Meanwhile from Figure 2 we can see that the standard deviations are within a reasonable margin for all three stor-

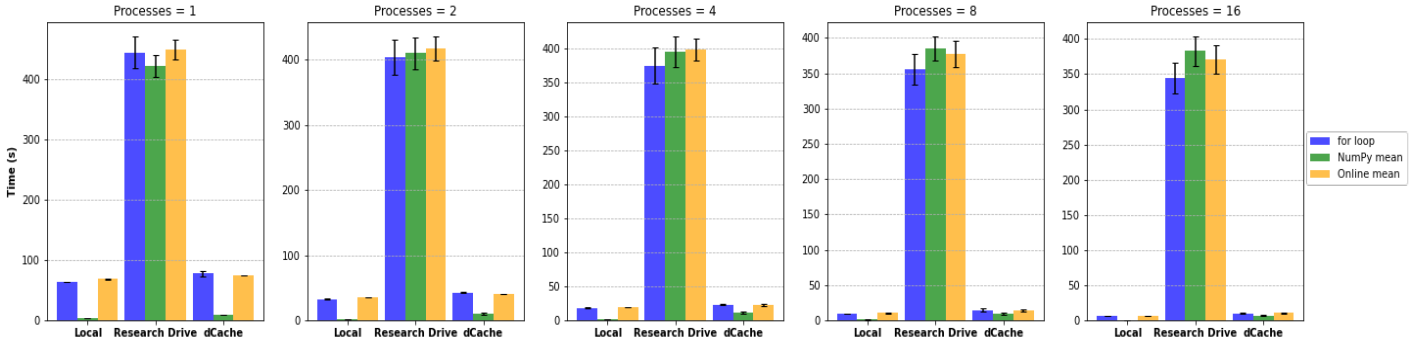


Figure 2: Graphical representations of the resulting mean time values and standard deviations for each storage system while changing the total number of processes.

Table 1: Mean time values and errors, in seconds, of each algorithm applied to data in three different storage types while changing the total number of processes.

	Alg./Proc.	1	2	4	8	16
Local	for loop	63.77 ± 0.07	32.81 ± 0.08	18.30 ± 0.40	9.51 ± 0.24	6.41 ± 0.39
	NumPy mean	2.73 ± 0.003	1.66 ± 0.04	1.07 ± 0.03	0.84 ± 0.08	0.63 ± 0.003
	Online mean	68.24 ± 0.20	35.11 ± 0.06	19.58 ± 0.08	10.15 ± 0.11	6.88 ± 0.05
dCache	for loop	77.38 ± 4.66	42.47 ± 1.19	23.23 ± 0.57	14.71 ± 1.82	9.97 ± 0.63
	NumPy mean	9.61 ± 0.35	9.93 ± 1.91	11.50 ± 2.00	9.51 ± 1.29	6.99 ± 0.98
	Online mean	74.26 ± 0.56	40.34 ± 0.50	23.06 ± 1.46	13.95 ± 1.07	10.15 ± 0.78
Research Drive	for loop	444.40 ± 26.48	403.77 ± 27.30	375.41 ± 26.70	355.46 ± 22.37	343.97 ± 21.94
	NumPy mean	421.64 ± 17.94	410.22 ± 24.00	395.57 ± 22.63	384.59 ± 17.23	382.79 ± 20.75
	Online mean	449.11 ± 16.59	417.14 ± 18.60	399.40 ± 16.30	377.19 ± 18.42	371.19 ± 20.05

age services. Only in the case of Research Drive are they slightly larger due to the unstable internet connection.

Conclusions

In this study we have tested three different storage systems under three different problems where we have shown that Research Drive stands as the slowest choice for handling stored data. By looking at the information in Table 1 one can see that, considering the "for loop" and "Online mean" algorithms, Research Drive is 7 times slower than the local storage when considering 1 process and about 50 times slower when considering 16 processes. For the "NumPy mean" algorithm, Research Drive shows to be 150 times slower to 600 times slower when compared to local storage considering 1 process and 16 processes, respectively. As for dCache, it has shown for the "for loop" and "Online mean" algorithms to be 1.2 times slower to 1.6 times slower when compared to local storage whereas for the "NumPy mean" algorithm it was 3.5 to 11 times slower considering 1 process and 16 processes, respectively.

Summing up, we conclude that when local storage (by our definition of it) does not have limitations of memory it is preferable for data processing. On the other hand, when dealing with large datasets, where the system's local memory is limited and the data is being shared with many users, dCache stands out as a strong candidate presenting very close results to the local storage.

For the final considerations, we have found that working with climate data is of great importance since allows people to trace models and make previsions about the future, but this can be a very demanding and resourceful task. By working with this kind of data and exploring which storage type allows to maximize the efficiency and speed of data processing, we are contributing to make this task less laborious and less expensive.

With regards to future work, it would be of interest to explore other datasets and extend this analysis to more storage services used in research such as iRODS, Ceph, CVMFS and OpenStack Swift.

References

[1] SURF Research Cloud documentation. (n.d.). SURF. Nl. Retrieved August 29, 2021, https://www.surf.nl/en/surf-research-cloud-collaboration-portal-for-research

[2] SURF Research Drive documentation. (n.d.). SURF. Nl. Retrieved August 29, 2021, from https://www.surf.nl/en/research-drive-securely-and-easily-store-and-share-research-data

[3] dCache.org. (2021, June 3). dCache. https://www.dcache.org/

[5] European Centre for Medium-Range Weather Forecasts. (2021, March 5). ERA5. ECMWF. Retrieved from https://www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era5

PRACE SoHPCProject Title

Maximising data processing efficiency in the cloud, with a twist for Research Data Management

PRACE SoHPCSite

SURF, Netherlands

PRACE SoHPCAuthors

Maria Li López Bautista, UB, Spain
João Guerreiro, FCT-UNL, Portugal

PRACE SoHPCMentor

Ander Astudillo, SURF, Netherlands
Arthur Newton, SURF, Netherlands
Carsten Schelp, SURF, Netherlands

PRACE SoHPCContact

Maria Li, López Bautista, UB
E-mail: mlopezba95@alumnes.ub.edu
João Q., Sério Guerreiro, FCT-UNL
E-mail:jq.guerreiro@campus.fct.unl.pt

PRACE SoHPCSoftware applied

Jupyter Notebook, Python, WebDAV

PRACE SoHPCAcknowledgement

We would like to express our most sincere gratitude to Ander Astudillo, Arthur Newton and Carsten Schelp for their continued support and guidance throughout the project. Also, great thanks to PRACE and the SoHPC organizers for making this amazing HPC journey possible regardless of the situation.

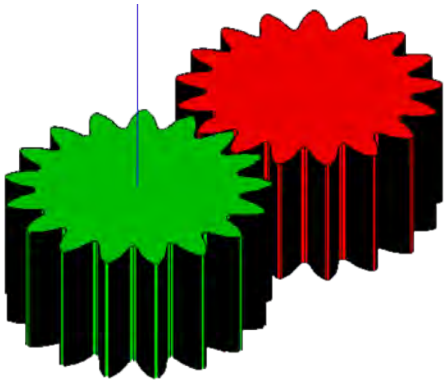
PRACE SoHPCProject ID

2127


Maria Li López Bautista


João Quintiliano Sério Guerreiro

Spur Gear Generation and Contact Analyses



Bartu Yaman, Ceren Tamkoç

Development of software that allows the user to easily produce a gear pair for finite element analysis and study the stress distribution of the contact region.

Spur gears are a type of cylindrical gears, having teeth that are parallel to the shafts. Spur gears are the easiest gear type to manufacture and use since they don't produce axial forces. Also they can work regardless of the number of teeth of the gear pairs and provide constant drive speed. Because of these traits, spur gears are the most commonly used gear type in various industries and this makes them valuable for analysis and studies.
Hence we are aiming to create a software to facilitate the analysis process of this type of gear by allowing researchers to automatically draw gears using their parameters.

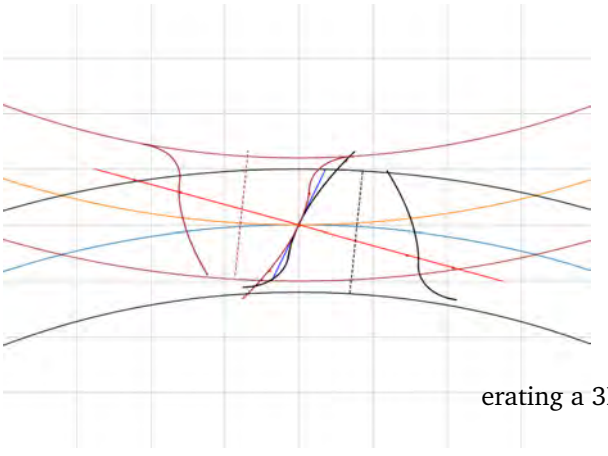


Figure 1: Spur gear geometry example.

How Do You Generate A Gear?

To generate a gear, we first need to calculate some points that will create the necessary curves once interpolated.
For this, we use the following equation of basic rack profile for spur gears to obtain the addendum part of the gear:

$$y_i = a_p.m.(1 - (1 - x_i/m)^n) \quad (1)$$

After that, we calculate the points that are on the pitch circle and that will connect the pitch circle body of the gear to the gear teeth with the following equations:

$$x_G = r_G sin\varphi_G \quad (2)$$

$$y_G = r_G cos\varphi_G - r_G \quad (3)$$

These calculated points are then checked for validity and rearranged if they are good to proceed with the drawing part of the procedure. Later, straight lines are drawn between the calculated points to join them together and obtain a completed gear profile on a 2D plane. This profile is then extruded by the user specified amount, thus generating a 3D gear model.

The same procedure is repeated for the second gear, which will complete the gear pair. The second gear is then translated in 2D into the correct orientation for analysis, producing the geometry shown above.

For this task, Python has been used to perform the necessary point/curve calculations and create a user friendly interface for the input values. For the drawing part, Python scripting for the SALOME software has been used.

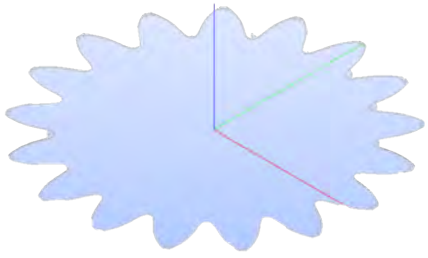
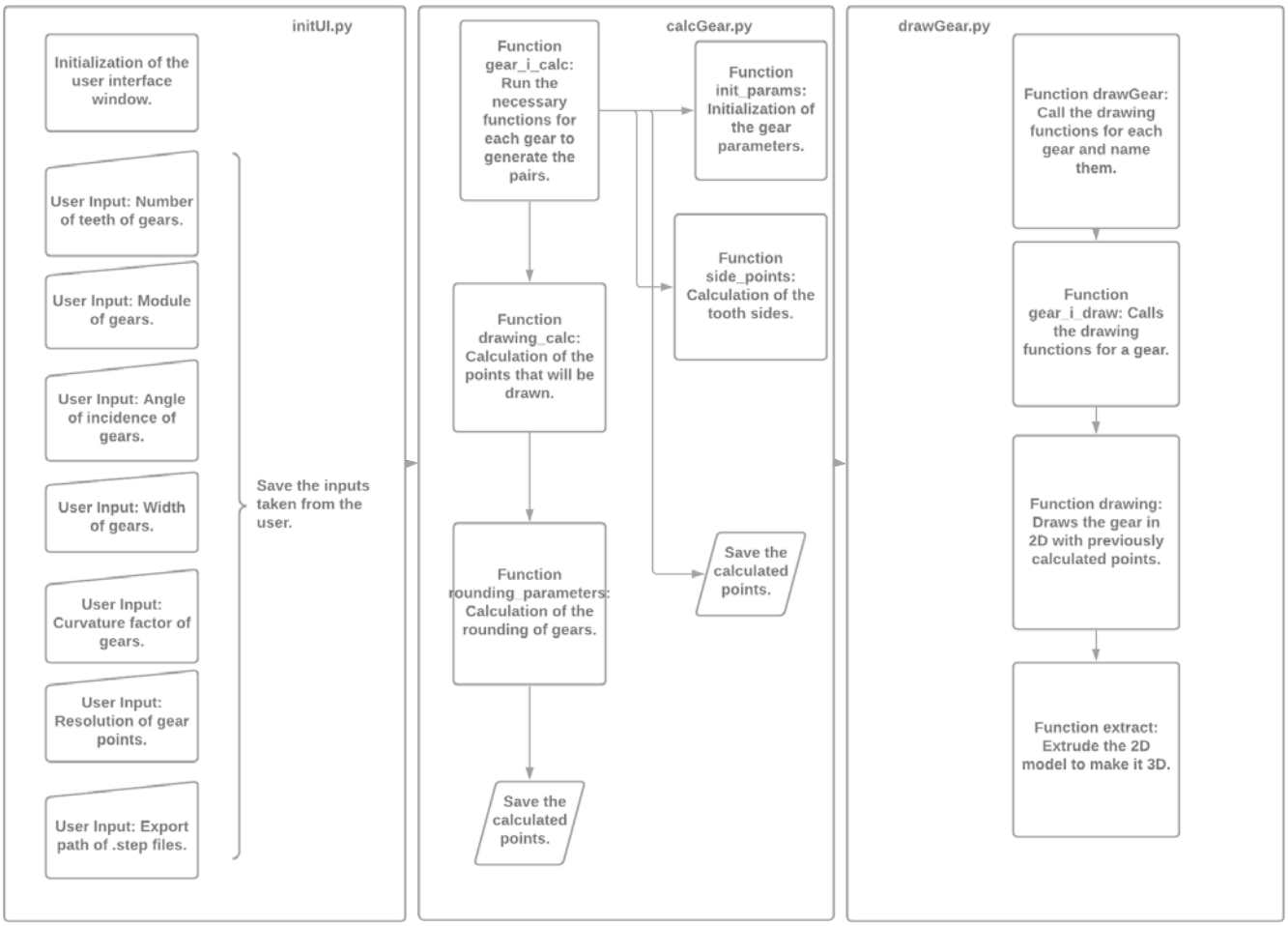


Figure 2: 2D geometry before extrusion into 3D.



Flowchart showing the function calls of the developed software

Software Design Choices

Our software consists of three separate files for user interface initialization, gear geometry calculation and drawing of the calculated points.

Parallel Computing?

Anytime HPC usage is mentioned, the first thing that comes to mind is parallel computing! However, parallelization in this context was not so simple. SALOME, the drawing software we used refused to work in parallel which forced us to split the procedure into two parts to at least parallelize where we can. Thus, we chose to parallelize the point calculations and engineered it in such a way that one thread calculates only one gear. This choice can also allow the software to scale better and easier in the future if it's development continues.

Not All Parameters Are Valid

The user may enter invalid gear parameters, which means that we need to handle the necessary errors. By design, we preferred to handle all the errors and carrying out all the necessary checks outside SALOME to save user time and effort. All of the checks are done during the point calculation process, allowing the user to immediately see the that there are invalid parameters and correct them before loading the parameters into SALOME. Invalid parameters are the only way this software can fail, therefore letting the user know about the errors outside SALOME makes the software very reliable and easier to use for the user since it takes some time for SALOME to reload the script which can be frustrating.

Save The Parameters

All of the necessary parameters and points needed to generate the gears are saved as compressed files inside the code directory. Which allows the user to redraw the last calculated gear pair without having to enter inputs again.

These choices add up to a reliable procedure for easy and fast gear pair generation, which can be used for contact analyses later on.

Finite Element Method

Finite element calculation method is in the form of processing the solution matrices transferred to the computer processor by transforming complex geometry and parts into a finite number of calculation cells called elements. The frequency of these calculation points affects the accuracy and precision of the solution. Demonstrating this sensitivity in studies is at least as important as validation. The general logic in this study is based on the determination of the values to which different element sizes converge, and the evaluation of working with the optimum element size independently of this element size. Since the finite element calculation method includes many sub-analyses such as different element sizes and different scenarios, the study using this method actually brings along many sub-studies. Ensuring all of these processes is possible with high-performance computers. The high number of processors and memory capacity (RAM) in these computers enable a wide range of analyzes to be performed quickly and effectively.

Meshing

Creating the Model Solid modeling is used to create the finite element mesh of gear. In order to control the geometric shape, number and density of the elements, the solid model is established as a collection of many regions.The points are used to define the lines and finally the surfaces are created by the appropriate combination of the lines. The parameters that determine the element number and density are applied to the edges of the regions. After these processes are completed, the gear finite element model is established with the meshing command. In the regions where the stress in the structure changes rapidly, element density is kept high, and in the regions where the stress changes more slowly, the element density is kept low. As a result, the finite element model of the gear obtained for the finite element stress analysis is shown in Figure 3.

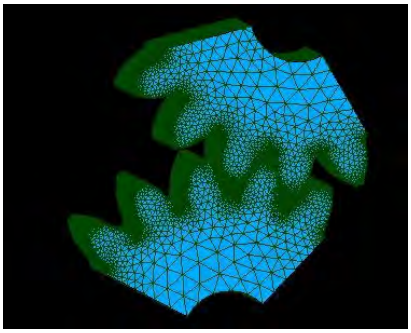


Figure 3: Meshing of Gear.

NETGEN 2D and 3D hypotheses were used to manage the NETGENPLUGIN parameters.
2D group allows defining the size of 2D elements

- Length from edges if checked in, size of 2D mesh elements is defined as an average mesh segment length for a given wire, else
- Max. Element Area specifies expected maximum element area for each 2D element.
- Allow Quadrangles - allows to generate quadrangle elements wherever possible.

3D groups allows defining the size of 3D elements.

- Length from faces if checked in, the area of sides of volumetric elements will be equal to an average area of 2D elements, else
- Max. Element Volume specifies expected maximum element volume of 3D elements.

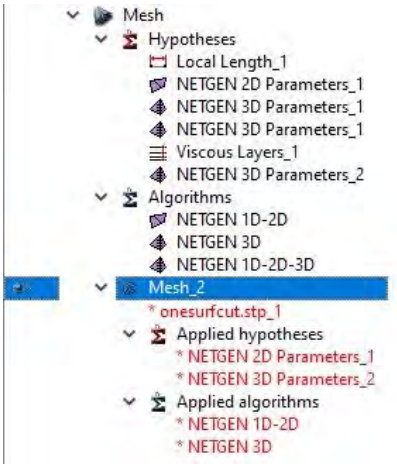


Figure 4: The algorithm and hypothesis .

The total element values created for element type and geometry are shown in figure 5.
1D egde : 8488 B32
2D faces: 163110 S6
3D volume: 1004492 C3D10

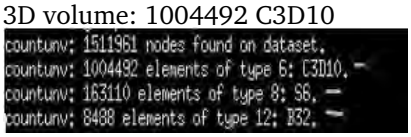


Figure 5: Element Types

1 Results and Discussion

As a result of the analysis by applying the boundary conditions, the physical behavior of the real structure is simulated.In order to determine the boundary conditions correctly, a subgroup has been formed in gear geometry.(Fig.6)

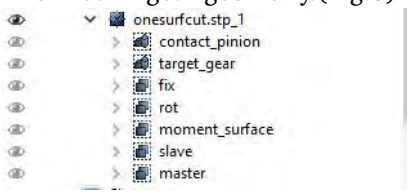


Figure 6: Groups from geometry

Boundary conditions are defined to specify the torque, rotation and supports which allow gear pair rotation about the Z-axis in the counterclockwise direction. Therefore, the torque of 0.9 Nm is applied at the gear shaft at Z-axis in that direction.The contact region of the gear is shown in Fig7. pinion is slave and gear is the master. Pinion material is POM; gear material is PA66.

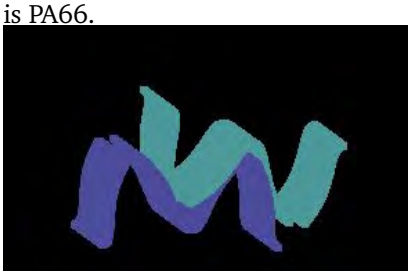


Figure 7: Groups from geometry

In the project, a static analysis was carried out to observe the stress distribution of the contact region of two spur gears. First, the geometry boundary condition was divided into groups in the Salome program, and then a mesh was created on the surface. After the mesh removal process was completed, after the boundary conditions were determined in the CalculiX, the solver was run and the results were obtained. According to the results of the study in the literature, the highest stress values were measured in the contact areas of the gear.

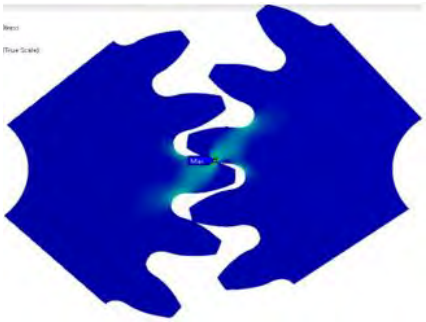


Figure 8: Maximum stress point

References

[1] Kulovec, S. K., &. Duhovnik, J. D. (2013). Variation of S-Gear shape and the influence of the main parameters.

[2] M. Raja Roy, S. Phani Kumar, D.S. Sai Ravi Kiran, "Contact pressure analysis of spur gear using FEA", International Journal of Advanced Engineering Applications, Vol.7, Iss.3, pp.27-41 (2014).

PRACE SoHPCProject Title

S-gear geometry generation and optimisation algorithm based on transient finite element mechanical/contact analyses

PRACE SoHPCSite

University Of Ljubljana, Slovenia

PRACE SoHPCAuthors

Bartu Yaman, Ceren Tamkoç, Middle East Technical University, Gazi University Turkey

PRACE SoHPCMentor

Borut Cerne, University Of Ljubljana, Slovenia

PRACE SoHPCContact

Bartu Yaman, Middle East Technical University
Phone: +90 530 462 4416
E-mail: yaman.bartu@metu.edu.tr
Ceren Tamkoç, Gazi University
Phone: +90 5072828791
E-mail: cerenntamkoc@gmail.com

Bartu Yaman

Ceren Tamkoç

PRACE SoHPCSoftware applied

SALOME, Python, Calculix, ANSYS

PRACE SoHPCMore Information

www.salome-platform.org/

PRACE SoHPCAcknowledgement

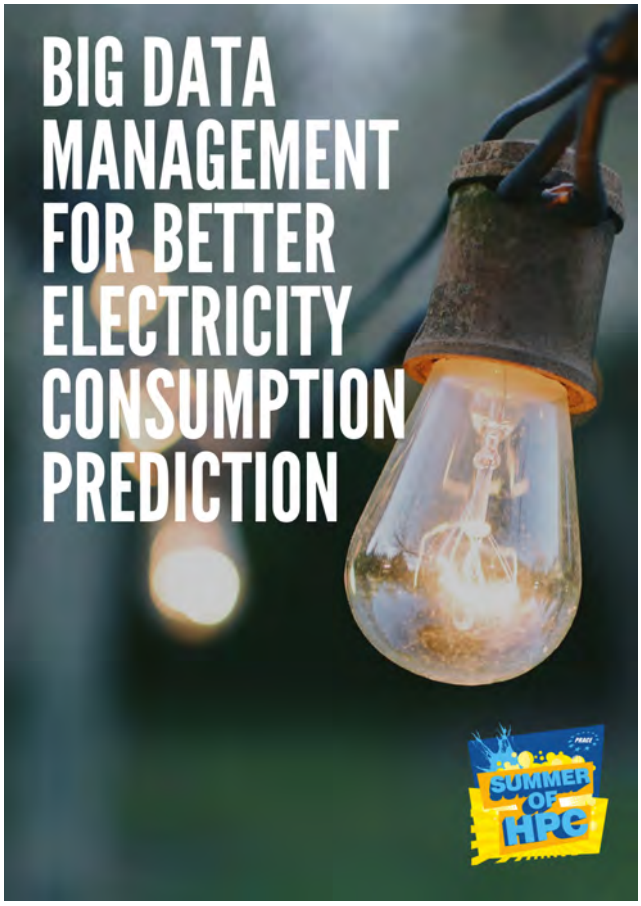
We would like to thank our mentor Borut Cerne for his time and effort.

PRACE SoHPCProject ID

2128

Big data Management for SoHPC

Irem Zeynep Dundar & Omar Patricio Pérez Znakar



Predicting energy consumption accurately is an important task for energy production. Therefore, the computation time is essential, since it will allow us to be able to produce the necessary energy for each section of the day and, therefore, not waste this essential element.

This project has been raised with the idea of testing how an existing code related to the prediction of energy consumption scales from a local server to a supercomputer.

This code has been developed with Python and R and performs the following actions: retrieve data, store it in MongoDB and reload it when necessary. In addition, based on historical data, we have developed scripts that build prediction models.

Therefore, the main objective of this project is to test how we can adapt the existing R and Python scripts so that we can build 10,000 models and predictions within the time limit of approx. 1 hour using a supercomputer with state-of-the-art storage and compute nodes.

1. Introduction.

This project will be oriented to verify how a code related to the forecast of energy consumption is scaled from a local server to a supercomputer. This process will be implemented through Python and R in order to perform different actions required to fulfill the objective of the project.

2. Data supplied.

The data used for the project has been provided by the professor in charge, we must clarify that this data is private, and therefore, the ID of the consumers has been encoded in such a way that it cannot be revealed who each consumer is. However, these data provide us with the necessary information for the prediction of energy consumption.

A. Test data.

To carry out the test, the data from one of the project files was required, so a small fragment of all the data was used for this. En concreto, tenemos 8*2=16 data files for 8 consumers, one file containing full historical data and one file containing data needed for prediction.

B. Final data.

Regarding the final data, we must clarify that this data has been collected through an external database (MongoDB) for this, the script provided has been modified to accept the integration online, resulting in all the data obtained throughout the project.To access this data a docker container running on one partition of the cluster was used (more information in the section 4).

3. Script used.

The sequential script used in this project was provided by the mentor, thus allowing the prediction of energy consumption. This process was possible thanks to small apprenticeships, constituting two categories: supervised learning and unsupervised learning.

A. Parallelization libraries.

To carry out this project, we have had to parallelize the code using various ex-

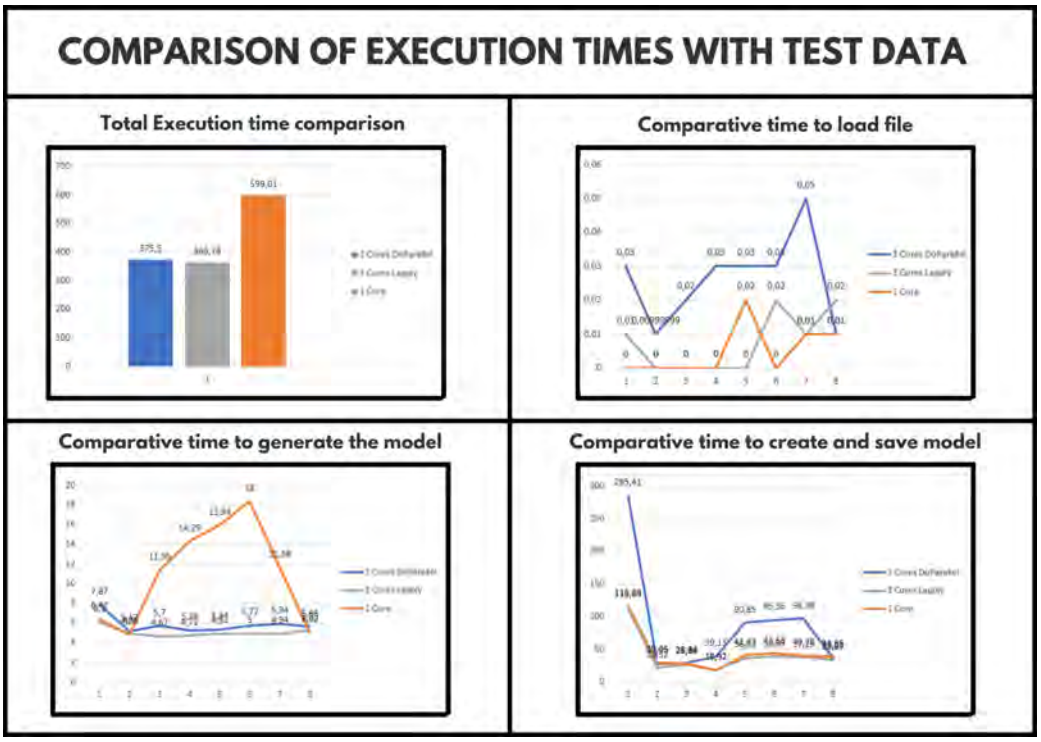


Figure 1: Comparison using test data.

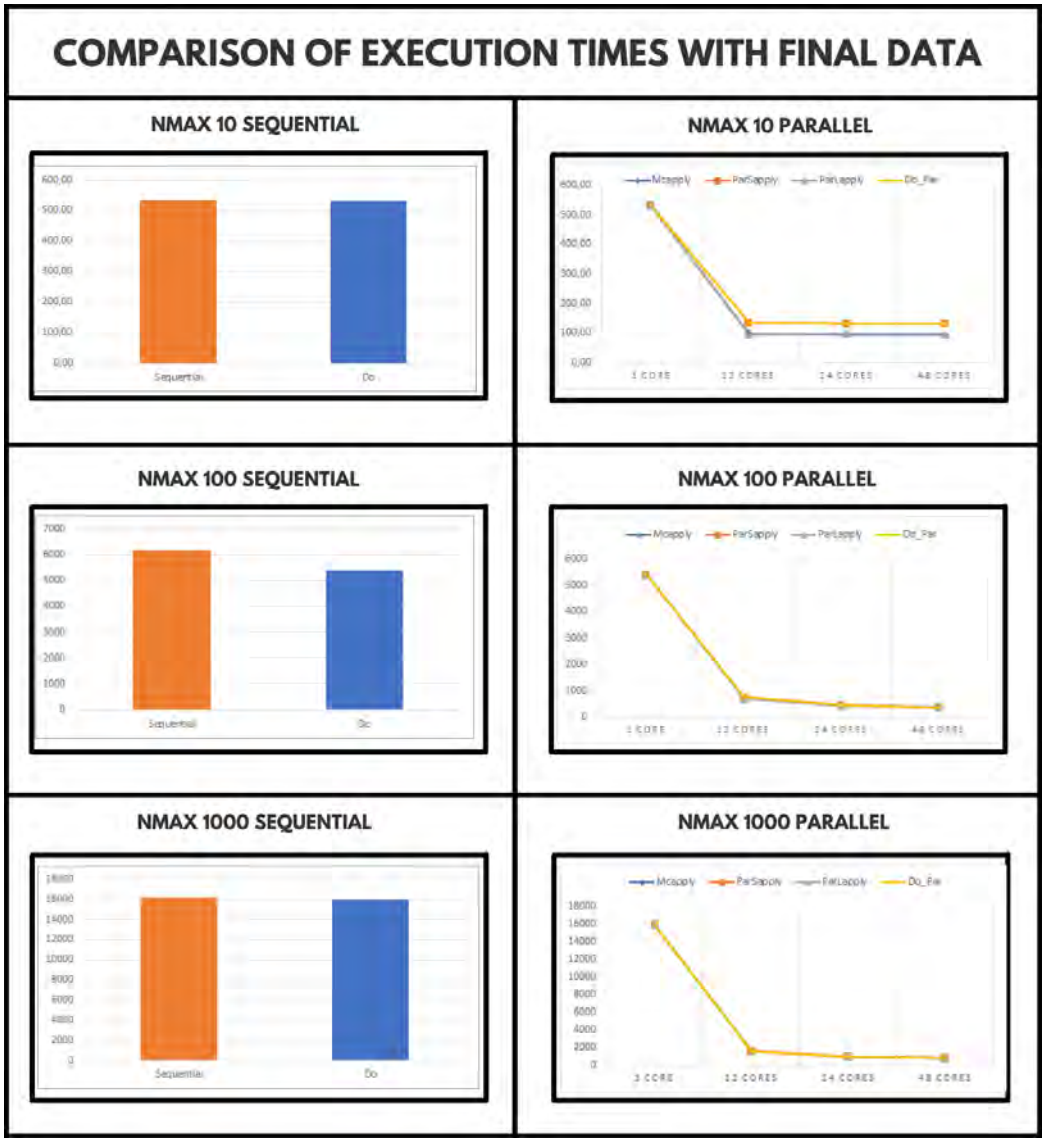


Figure 2: Comparison using end data (MongoDB).

isting libraries. For this reason, some of the most important libraries have been chosen and incorporated into the code in order to later make a comparison and select the most appropriate one. The libraries used can be seen below:

- **DoParrallel, foreach and Parrallel[4]:** these three libraries allow us to be able to parallelize a code without doing much configuration.
- **Mcapply [1,2,3]:** It is one of the simplest ways to parallelize the code and, therefore, one of the most commonly used.
- **ParSapply [1,2,3]:** It is a parallel function to the "sapply" method, allowing to create a cluster from where the code can be executed.
- **ParLapply [1,2,3]:** It is a function that allows us to use the "lapply" method in parallel. In adition, it allows us to execute the code in a cluster (previously created).

4. Server used.

The main objective of the experimentation has been to be able to measure times through the various ways of parallelizing the code to select the most suitable library for this purpose.

A. Server used to run the scripts.

To carry out the project, the mentor in charge provided us with a server, which already had the R program incorporated, which in turn expedited the work by not having to require installation and maintenance tasks. In addition, this server has a number of 48 cores, which opens the way to a much more in-depth comparison, since a greater number of cores does not always imply less task execution time.

B. Server used for MongoDB container.

For this section, a MongoDB container has been created (via a Docker container) that has the final project data stored. Likewise, a REST API has been made to be able to contact this database. This REST API has been created using Python (using the Flash library). This procedure is used to interact more easily and to facilitate access to external data.

5. Experimental results of the performance of each library.

The main objective of the experimentation has been to be able to measuretimes through the various ways of parallelizing the code, to selectthe most suitable library for this pur-pose.

A. Local Computer with test data.

For the development of a first comparison locally, we have used the test data inside a conventional computer (3 cores) using the modified code with the libraries that allow us to parallelize. We can see the result obtained in Figure 1.

As we can see in the previous Figure 1, the highest execution time obtained is the sequential version (of course). Similarly, it should be noted that the times obtained in each parallel version are more or less similar, therefore, we should not opt for any, at least for now.

B. Server with final data.

For the development of the comparison, the server provided by the research mentor has been used. By having 48 cores, the code has been executed using 48, 24 and 12 cores to obtain more complete results. It should also be noted that the code has been modified by adding a new variable called "Nmax". This allows us to configure the size of the problem and, therefore, through logic, the smaller it is, we should obtain a shorter execution time.

After carrying out the comparison (Figure 2), we have detected that for a medium problem size (Nmax 10), the most optimal is to use 12 cores since they provide us with less execution time and we can leave the other 26 cores available for another execution, however, for a larger problem size (Nmax 50), we can see that using 48 cores is the optimum option.

Finally, we can say that ParSappply obtains a shorter execution time, however, it should be noted that ParLapply has a higher time but quite similar, therefore, the most logical thing would be to use either of these two options.

6. Conclusions.

To explain the conclusions, we will take into account the following points:

- **First step to take:** one of the most important steps is to fully understand the data and what is required with it.
- **Technology comparison:** there are several technologies to parallelize the code, however we need to do some research to see which one best suits our problem.
- **Number of cores:** not always, a greater number of cores gives us a higher processing speed. This is because the joining time of these can be greater than the advantage that having more cores gives us.
- **Fundamentals of power consumption measurement:** for a smaller problem, it is better to use 12 cores; however, for a large problem, the best option would be to use 48 cores.
- **Best technology:** With a large amount of data, all technologies give us the same results.

References

[1] ClusterApply: Apply Operations using Clusters https://www.rdocumentation.org/packages/parallel/versions/3.6.2/topics/clusterApply

[2] Quick Intro to Parallel Computing in R https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html

[3] Parallel Computation https://bookdown.org/rdpeng/rprogdatascience/parallel-computation.html

[4] A guide to parallelism in R https://privefl.github.io/blog/a-guide-to-parallelism-in-r/

PRACE SoHPCProject Title

Big data management for better electricity consumption prediction.

PRACE SoHPCSite

University of Ljubljana, Slovenia

PRACE SoHPCAuthors

Irem Zeynep Dundar [Koç University, İstanbul], Omar Patricio Pérez Znakar [University of La Laguna, Spain]

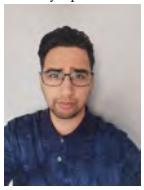
PRACE SoHPCMentor

Janez Povh, University of Ljubljana, Slovenia

PRACE SoHPCContact

E-mail: leon.kos@lecad.fs.uni-lj.si

Irem Zeynep Dundar



Omar Patricio Pérez Znakar

PRACE SoHPCAcknowledgement

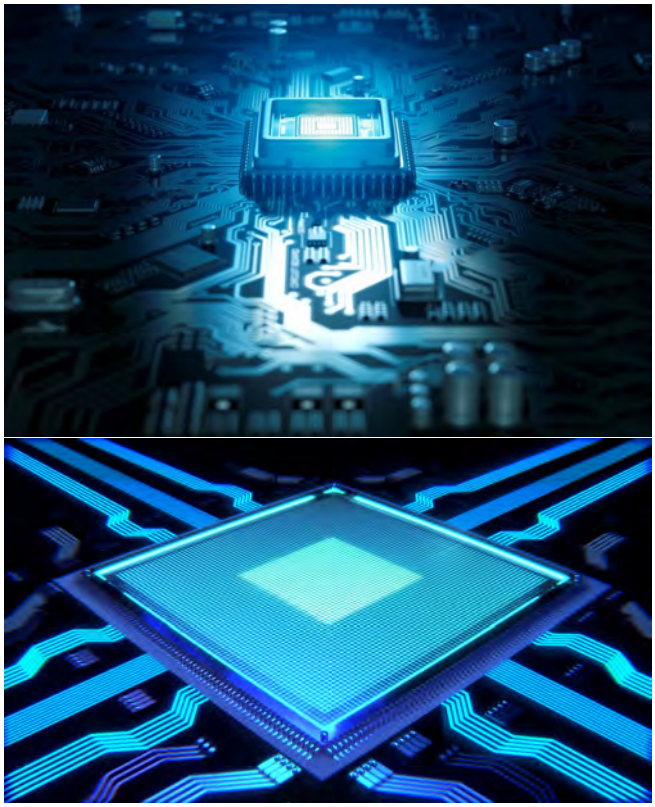
I would like to end this document by thanking both Professor Janez Povh and Matic Rogar for the wonderful job they have done. Both have been a great support for us on this occasion, remaining fully available for meetings and doubts at all times and even, on several occasions, adapting to the students in order to have a better experience.

PRACE SoHPCProject ID

2129

Making TopOpt run faster using GPUs

Martin Stodůlka, Theodoros Aslanidis



Lately, GPUs have become ubiquitous in HPC computing and used as accelerators for numerous applications. Therefore, it is only logical that scientific applications can take advantage of their speed and versatility too. Our project aims to reduce the execution time of the Topology Optimization problem while leaving the accuracy of the results intact.

The Topology Optimization problem is a mathematical method which spatially optimizes the distribution of material within a defined domain, by fulfilling given constraints previously established and minimizing a predefined cost function. When we want to apply Topology Optimization to an object or an element we can separate the process into three stages. At first, we define a material block and we discretize it in elementary blocks. According to a list of objectives, topology optimization determines if we need or not this elementary block. The sum of these elementary results gives us a proposal of the shape. The scope of this application can be aerospace, mechanical, biochemical, and civil engineering. It can lead to better performing solutions in drastically shorter amount of time and, hence, reduced costs. A topology optimization problem can be written in the general form of an optimization problem as:

$$\underset{\rho}{\text{minimize}}\ F = F(u(\rho), \rho) = \int_{\Omega} f(u(\rho), \rho) dV$$

$$\text{subject to } G_0(\rho) = \int_{\Omega} \rho dV - V_0 \leq 0$$

$$G_j(u(\rho), \rho \leq 0 \text{ with } j = 1, \ldots, m$$

Original Implementation

The original TopOpt[1] code given to us as a starting point is a console application used for large-scale topology optimization on structured grids. It's written in C++ and makes use of the PETSc[2] library that is responsible for all the calculations. Moreover, there are some python scripts used for converting output *.bin* files to the *.vtu* file format that ParaView takes as an input to illustrate the final result.

Thanks to the PETSc library, TopOpt is inherently able to run on multiple CPUs on multiple nodes. It mainly uses solvers for the system of linear equations (KSP) from PETSc for its computation. The console application itself is flexible, allowing anyone skilled enough in programming to alter the optimization or write their own using their classes like TopOpt, Physics, Filter, etc.

The base implementation without any changes outputs a cantilever beam which can be used to test any changes. Our main task was to change the TopOpt code to work on GPUs in hopes

of achieving better performance than the original CPU implementation due to the inherent massive parallelism of GPUs. All measurements and tests were done on the ULHPC supercomputer Iris.

TopOpt Output Examples

Figure 1 depicts six optimized cantilever beams on a $2 \times 1 \times 1$ domain allowing 12% material discretized by 27.6 million elements. The design problems differ through the filter radius. The figure also illustrates the internal holes generated for the smaller filter radius. Through the above image, one can without difficulty notice that the result given by the TopOpt application is not something fixed but can vary greatly depending on the parameters set. The parameters usually are influenced by the goal we want to achieve, which is different from field to field.

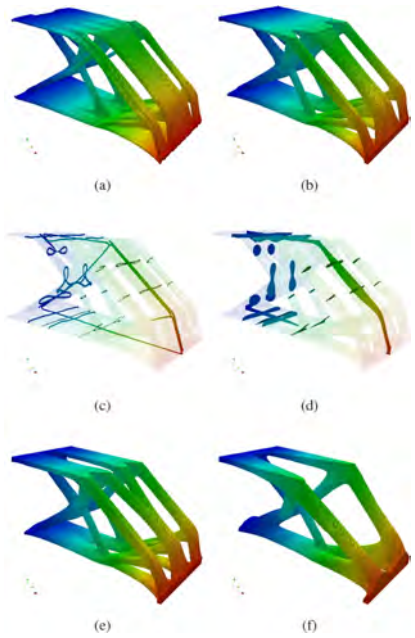


Figure 1: Different cantilever beam TopOpt solutions

PETSc

PETSc is a library specially designed for mathematical operations modeled by differential equations. It supports MPI for parallelism across multiple nodes as well as CUDA/OpenCL for GPUs and hybrid CUDA/MPI for GPUs across multiple nodes.

It consists of multiple modules/classes. Each class is used for one type of operation/data structure such as: Matrices, KSP, SNES etc. We will be focusing on KSP, because of the results from profiling of the code mentioned further down below.

KSP is a package for linear system solvers using Krylov subspace method and a preconditioner. Krylov subspace method is used for solving linear systems using vector multiplied by matrix instead of matrix-matrix multiplication. Preconditioning is used to speed up the convergence of Krylov methods.

Profiling

The first step before attempting any sort of optimization was to identify slowest parts of code and setup a baseline for speeds and scaling of current implementation. We used VTune to get a complete picture of where most of the time is spent.

After running the original implementation of TopOpt on 4 nodes we have generated and inspected the profiler output. As we can see in the Fig. 2 below, KSPSolver is taking up 80% of computational time. KSPSolve is a function from the PETSc library that solves linear system of equations.

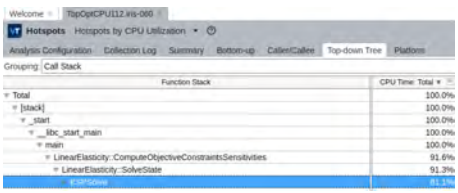


Figure 2: Profiling of the TopOpt application on a single node using the VTune tool

From our results TopOpt scaled only up to 4 nodes after which there was minimal benefit to adding more compute nodes. The main reason for that is because the critical parameter was the MPI processes that the program uses. Beyond a certain level, no matter how many nodes we add, the execution time will not decrease unless we increase the MPI processes. But this is something we can not increase as much as we want, because it depends on how strong (how many cores and therefore threads) each node has. From 1 to 4 nodes when we increase the MPI processes, the application seems to be able to take advantage of the resources given to it to the fullest, but from then on, this is something that does not apply. The table 1 contains all the times we measured during our experiments.

CPU variant of TopOpt

Number of Nodes	Time(s)
1	270
2	199
4	171

Table 1: TopOpt scaling with nodes

Using GPUs with PETSc

PETSc has an easy way to port existing code over to using GPUs. There are no special GPU functions for already existing CPU functions or any other additional functions that need to be called to manage memory on GPUs. Instead, all matrices and vectors must be of a special CUDA type, and then PETSc will automatically execute any operations using CUDA variants of arguments (if they are supported) on GPU. To be more specific, let V and M be a vector and a matrix respectively, then the following functions should be used:

VecSetType(V, VECCUDA)
MatSetType(M, MATAIJCUSPARSE)

This way, any operation related to the above will be done on the GPU and the result will then be transferred to the CPU.
However, the existing implementation of TopOpt did not work by simply switching types of matrices and vectors to CUDA type. It turns out that certain combinations of KSP solvers and preconditioners do not work with GPUs. The preconditioner method that the original CPU implementation was using is called PCMG (Multigrid Preconditioning). This specific method had not an immediate GPU support, so we had to try different preconditioners. We tested all combinations of two solvers and three preconditioners for GPUs that we have found based on our research through similar libraries that use GPUs with PETSc, to solve a linear system of equations. We have chosen GMRES and PCG as our solvers and NONE (Incomplete LU), GAMG and MG as our preconditioners. We also tried to setup their parameters by reading from documentation pages of PETSc.

Maintaining correctness

When we finally had our first successful execution of the program using the

GPU, our first concern was to make sure that our implementation did not alter the original CPU result and the accuracy of the program remained the same. To find out, we used ParaView to see the output of TopOpt both in the initial implementation and in our own.

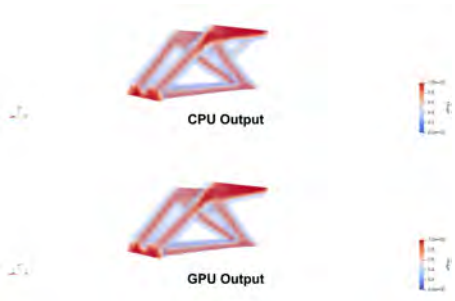


Figure 3: CPU and GPU TopOpt output comparison using ParaView

As shown in the figure 3 above, the two results were identical, so we proceeded to the next stage which was the comparison of the performance of the two implementations.

Results

We tested many different parameters to see their effect on the execution time. Some of them are the number of nodes, the number of MPI processes, GPUs, as well as solvers, and preconditioner types. We tested all combinations of KSP solvers and preconditioners and we observed that GMRES solver and ILU (default) preconditioner was the only one that outperforms the original implementation by a small margin. With 8 GPUs (2 nodes), the implementation was about 15 seconds faster or 1.11 times faster. Fig. 4 shows the time comparison between the two implementations.

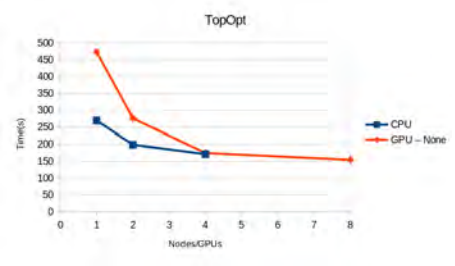


Figure 4: Time Comparison of CPU and GPU variants of TopOpt (in nodes and GPUs respectively)

Another series of experiments that we tested in the GPU variant, showed that the different topology used in the resources for the execution of the application does not play an important role in the execution time. More specifically, if the number of GPUs is the same, the execution time is similar whether the GPUs are on the same node or a different node. Even in the extreme case where we tried to run the program using a total of 4 nodes and 1 GPU per node, the execution time was similar to the case where we used 4 GPUs in 1 node. In conclusion, the parameter that plays the dominant role in the program's performance is the number of MPI processes. As in the case of the CPU implementation, the GPU one also does not show any performance improvement if we use more than 4 GPUs. The maximum number of MPI processes that Iris clusters can support is 4, so that was our limit.

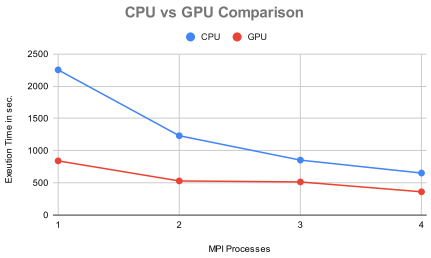


Figure 5: CPU vs GPU comparison as we change the number of MPI processes

Fig. 5 shown above shows that our effort is outperforming the original CPU code for 1, 2 3, and 4 MPI processes. This result is quite satisfactory and encouraging.

Conclusion and Future Work

In this article, we explained what Topology Optimization is, and analyzed the methodology we followed, to improve its performance. This process was not straightforward, as various obstacles troubled us. We proved that such an effort has positive results as our code has less execution time than the original. Future work could be a further analysis of the preconditioner methods and the attempt to properly configure them through the PETSc library so they can also run on GPUs. Preconditioner methods such as multigrid that performed well on the CPU may improve the performance of the GPU even further.

References

[1] Niels Aage, Erik Andreassen, and Boyan Stefanov Lazarov. Topology optimization using petsc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization*, 51(3):565–572, Mar 2015.

[2] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, 2018 Victor Eijkhout 45 1.7. DIRECTORY STRUCTURE PETSc 3.10 September 20, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc Web page, 2018. http://www.mcs.anl.gov/petsc.

PRACE SoHPCProject Title

Designing Scientific Applications on GPUs

PRACE SoHPCSite

ULux-University of Luxembourg, Luxembourg

PRACE SoHPCAuthors

Martin Stodůlka,
Theodoros Aslanidis,

PRACE SoHPCMentor

Dr. Ezhilmathi Krishnasamy

PRACE SoHPCContact

Martin Stodůlka, Brno University of Technology
E-mail: stomartin12@gmail.com
Theodoros Aslanidis, University of Thessaly
E-mail: teodorico.asla@gmail.com

PRACE SoHPCSoftware applied

PETSc, Paraview, VTune

PRACE SoHPCMore Information

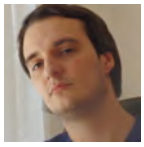
PETSc, ParaView, VTune

PRACE SoHPCAcknowledgement

We would like to express our gratitude to our mentor Ezhilmathi Krishnasamy providing us the guidance throughout the project. We would also like to thank PRACE for organizing the Summer of HPC event and giving us the opportunity to spend our summer so creatively.

PRACE SoHPCProject ID

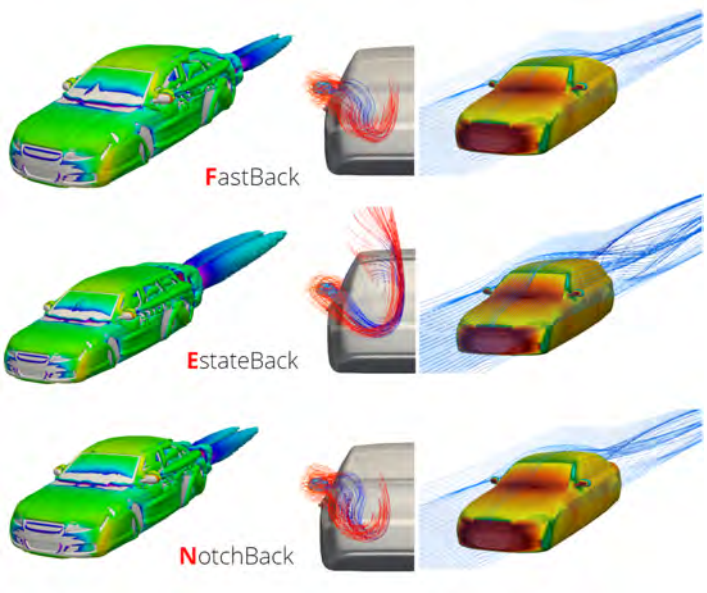
2130


Martin Stodůlka


Theodoros Aslanidis

DrivAer car aerodynamics

Paolo Scuderi and Benet Eiximeno Franch



The aim of this project was learning how to perform a CFD analysis of a car and understand which of the most common car geometries had the best performance from an aerodynamic point of view.

The DrivAer is a generic car model developed by the Technische Universität von München. They provide several CAD models for different configurations. All the configurations are based on three different car geometries (Fastback, Notchback and Estateback) and for each cases there are various CAD models with increasing level of detail. The most basic models only include the shape of the body without wheels and mirrors, while the most elaborated also have the design of the internal components such as the radiators, engine and exhaust.

During the development of this project, the three different geometries have been used, always working with the configuration that has side rear view mirrors but not wheels nor internal components, on account of the good accuracy given by analyzing the body when comparing the drag formation mechanisms of cars.

Although the comparison of the three geometries is important, in CFD it is also important to compare the setups of the cases in order to see how they affect the results. Therefore, besides the comparison between the three geometries a deep analysis of how the case configuration changes (i.e. solver, turbulence model, and geometry) affect the result has been done.

Mesh assessment and grid sensitivity

In order to find the appropriate mesh strategy and sizes, three tests have been conducted making use of the Fastback geometry at $Re = 9 \times 10^6$ which is the highest to be analyzed. All of them combining extruded triangles in the boundary layer region and tethraedra in the rest of the domain.

The size of the domain is the one specified in Heft et. al.[1] It has remained unchanged in all three cases. The longitudinal axis of the car is parallel to the X axis, which has the direction of the inlet airflow. The Z axis is oriented vertically, starting from the road plane and the Y axis forms a positive base with the other two.

- Version 1: 8 million elements.
- Version 2: 46 million elements.
- Version 3: 72 million elements.

It should be noted down that the version 3 mesh had too high non-orthogonality levels in the boundary layer elements located in the side rear-view mirror area due to the geometry constraints and it was not possible to run it during the Summer. Hence, only mesh version 1 and 2 have been used for this study. Figure 1 shows the second version.

Figure 1: Mesh 2.

Figure 2 illustrates the comparison of the C_P distribution on the symmetry plane of the car of the two meshes with the experimental data obtained by Heft et. al.[1] at the same Reynolds number. Although in this model the wheels have not been considered, the results show a clear agreement between the method and the experimental results for the flow near the wall, however, figure 3 shows a big change in the value of

the distribution of the upstream velocity in the wake of the car between the two meshes. It is clear that the results discrepancy between the two meshes relies on the different wake resolution, hence, the version 2 of the mesh will be taken as it is the one which provides more accurate results.

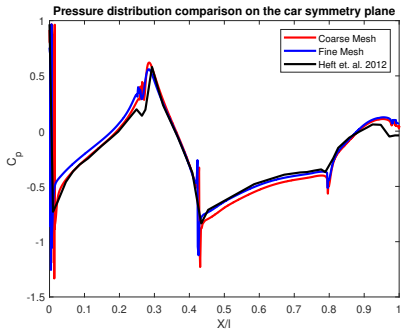


Figure 2: Pressure distribution on the car symmetry plane.

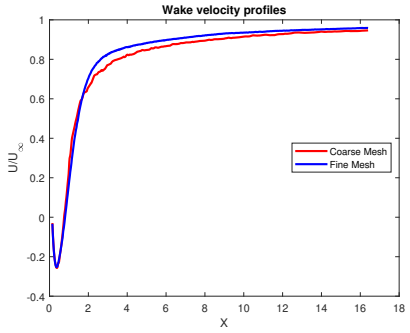


Figure 3: Wake velocity profiles.

Case setup

Considering the time and resources available, all of the simulations have been performed using Reynolds Averaged Navier-Stokes turbulence models with a wall model in the boundary layer area. This made feasible to run every mentioned simulation mentioned simulations in time getting a good accuracy in order to understand the flow structures around a car.

The boundary conditions for velocity and pressure in the different regions are the following:

- Inlet: fixed velocity value with zero pressure gradient.
- Outlet: uniform velocity value and fixed pressure value.
- Sides: slip wall with zero pressure gradient.
- Car: no slip wall with zero pressure gradient.
- Road: fixed velocity value with zero pressure gradient.

The steady state simulations have been computed using the simpleFoam solver, while the transient simulations have been performed with pimpleFoam taking a maximum Courant number of 1 and an adjustable timestep in order to meet the CFL stability condition.

Finally, the case has been decomposed in order to run it in the HPC machine Iris of the University of Luxemburg. All cases have been run with 672 cores, which is a number that allowed to run the cases in a reasonable time without excessive long waits in the queue.

The paralelization of the cases is done by splitting the mesh in between the available processors. This division can be obtained using several algorithms and it has a huge impact on the scalability and performance of the computation. In this project two of them have been tested in a 46 million elements mesh:

- Simple: it splits the domain volume equally between the processors.
- Metis: optimized algorithm to assign to each processor the same amount of elements with the minimum amount of nodes shared between them.

The performance impact between both algorithms is quite significant, being Metis the fastest option taking no more than 1.5 seconds per iteration while using the decomposition made with the Simple algorithm it takes more than 3 seconds to do the same computation.

Results

Several simulations were run to investigate the effects of the geometry in the flow structures and the impact of the changing in the Reynolds number and the turbulence models. Starting from a converged solution, the previously exposed boundary conditions have been imposed. The simpleFoam algorithm, already implemented in OpenFOAM, was used. The following incompressible Navier-Stokes equations were solved. $\forall i, j \in \{1, ..., d\}$

$$\partial_{x_i} u_i = 0$$
$$\partial_t u_i + u_j \partial_{x_j} u_i + \rho^{-1} \partial_{x_i} p - \nu \partial^2_{x_j^2} u_i = f_i$$

where d is the dimension of the problem.

Effects of the geometries

To investigate the effects of geometry in the flow structure, the following values have been imposed: $U_\infty = 10$ m/s and the kinematic viscosity as $\nu = 10^{-6}$. These fictitious values are commonly used in the majority of OpenFOAM tutorials. The simulations were run with the already mentioned criteria on the ULux cluster. To post-process the results, the attention was focused on the coefficient of pressure and on the computation of the Q-criterion. Starting from the definition of the C_P for incompressible flows

$$C_P = 1 - \left(\frac{U}{U_\infty}\right)^2$$

where U is the local velocity magnitude, from figure 4 is observed similar front flow structures for all the geometries. The rear body changes the backflow structures.

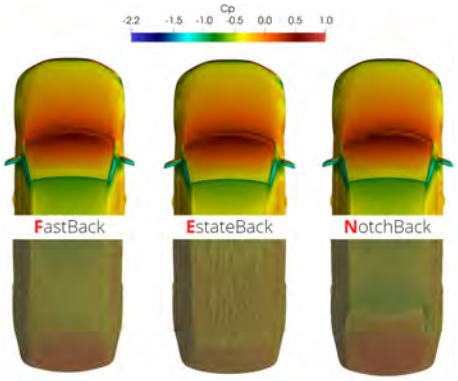


Figure 4: Coeffients of pressure distribution and Line Integral Convolution in the back.

In particular, for the Fastback and the Notchback configurations, the flow starts to decrease its velocity magnitude. This decrement involves an increment in the pressure value, i.e. adverse pressure gradient. Therefore, these will be areas of possible fluid separations. It is more important in the Notchback shape, where the presence of the rear horizontal compartment implies a more significant deceleration of the fluid flow. For these reasons, in both the geometries, figure 4 shows the presence of two recirculation bubbles, which are not present in the Estateback configuration. These bubbles affect the flow structure in the wake region. For the Estateback configuration, the wake vortices have bigger width than for the other two configurations, as shown in figure 5.

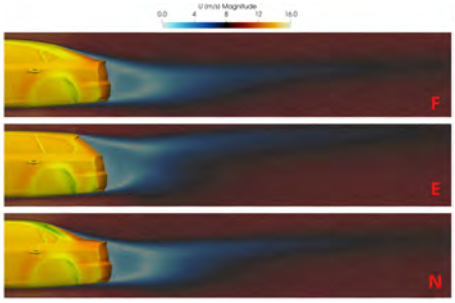


Figure 5: Back recirculation bubbles in XZ plane.

Introducing the definition of the scalar Q-value according to Hussain[2]

$$Q = \frac{1}{2}\left[\mathrm{tr}\left(\underline{\mathbf{\Omega}}\underline{\mathbf{\Omega}}^T\right) - \mathrm{tr}\left(\underline{\mathbf{S}}\underline{\mathbf{S}}^T\right)\right]$$

where $\underline{\mathbf{\Omega}}$ is antisymmetric and $\underline{\mathbf{S}}$ is the symmetric part of the decomposition of the velocity gradient $\nabla\mathbf{u}$. When the scalar Q-value is positive areas of higher vorticity than strain rate in the flow are being considered. Obviously, if it is null the two values are perfectly balanced. In particular, the Q-criterion computed for $Q = 10$ in figure 6 shows the turbulence structures for the N geometry. Look at Scuderi and Eiximeno for F and E comparisons.

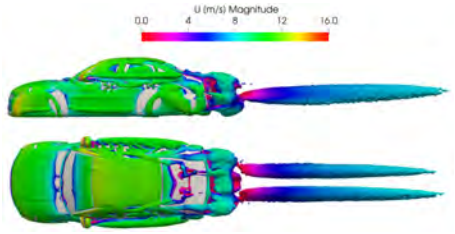


Figure 6: Q-criterion. Wake structures for Notchback.

Effects of the Reynolds number

To analyze the effects of the Reynolds number, according to the capability of OpenFOAM to work with the dimensionless equations, without changing the velocity boundary conditions, several values of kinematic viscosity were chosen to have the following Reynolds number: 3×10^6, 6×10^6, 9×10^6. In particular, having fixed the velocity $U_\infty = 10$ m/s and the reference length as $L = 4.5$ m, respectively the following kinematic viscosities have been set: $\nu = 1.5 \times 10^{-5}$ m^2/s, $\nu = 7.5 \times 10^{-6}$ m^2/s, and $\nu = 5 \times 10^{-6}$ m^2/s. All of the simulations were run for the three geometries. The results were computed in terms of C_D values as shown in figure 7. The three distinct lines are typical for a blunt body after the so-called subsonic crisis of resistance. Generally speaking, cars are blunt bodies hence the biggest contribution to the aerodynamics forces is associated with pressure.

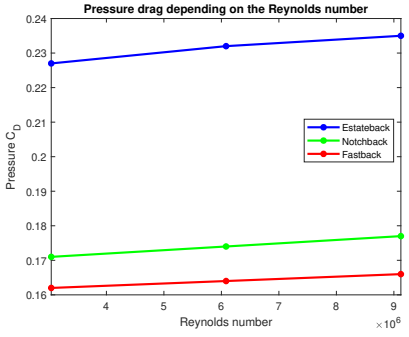


Figure 7: Dimensionless drag comparison.

Effects of the turbulence models

To investigate the effects of the turbulence models the Fastback configuration has been used setting inlet boundary conditions velocity as $U_\infty = 10$ m/s, kinematic viscosity as $\nu = 10^{-6}$ m^2/s, and the previously reported parameters. Two equations models were used: $k-\epsilon$, $k-\omega$, and $k-\omega\ SST$. The dimensionless drag values were computed respectively, $C_D = 0.130$, $C_D = 0.824$, and $C_D = 0.162$. The most accurate is the $k-\omega\ SST$. The $k-\omega$ estimates accurately the flow near the wall. In contrast, the $k-\epsilon$ predicts well the flow far from the wall. Both the effects are taken into account thanks to the $k-\omega\ SST$. Consequently, slicing along the X axis the internal mesh used to compute the wake region, we point out a lot of discrepancies for the $k-\omega$. On the contrary, the $k-\epsilon$ wake results are in line with the k-Omega SST ones.

Future works

Future works will be based on the increment of the complexity of the car and the numerical models. More detailed numerical results can be obtained by using DES and LES starting again from cars without wheels. Then the introduction of the wheels is mandatory before going on with all the other details.

Conclusions

The work aims to investigate the external flow over three different no-wheels rear back models of the DrivAer car. On our local machines the meshes for all three configurations were prepared: Fastback, Estateback, and Notchback. The mesh accuracy was tested by comparing our numerical wake results with the experimental one provided by Helf et al.[1] On the ULux cluster OpenFOAM was used to compute the numerical solution based on RANS. Three different comparisons were made: the first one purposes the effects of the geometry in the flow structures. The second one has been based on the study of the changing in the Reynolds number for each geometry configuration. The third one aims to understand the effects of turbulence models in the numerical solution for the Fastback version. In the end, the Estateback configuration has the biggest C_D value, hence the most considerable turbulence structures in the wake region. No huge variations in the C_D values are present for a selected configuration. This is typical of all the blunt body in the fluid dynamic regime studied. The most accurate turbulent model tested is the $k-\omega\ SST$. In the wake region, the $k-\epsilon$ results are not so far from it.

References

[1] Heft et. al. (2009). Experimental and numerical investigation of the DrivAer model.

[2] Hussain, F. (1995). On the identification of a vortex.

PRACE SoHPCProject Title

Aerodynamics

PRACE SoHPCSite

High Performance Computing - Université du Luxembourg, Luxemburg

PRACE SoHPCAuthors

Paolo Scuderi,
Politecnico di Torino, Italy
von Karman Institute for Fluid Dynamics, Belgium
Benet Eiximeno Franch,
Universitat Politècnica de Catalunya BarcelonaTech, Spain

Paolo Scuderi

Benet Eiximeno Franch

PRACE SoHPCMentor

Ezhilmathi Krishnasamy, ULux

PRACE SoHPCContact

Ezhilmathi Krishnasamy, Université du Luxembourg - Maison du Nombre 6, Avenue de la Fonte L-4364 Esch-sur-Alzette - Campus office: MNO, E02 0215-050
Phone: (+352) 46 66 44 6265
E-mail: ezhilmathi.krishnasamy@uni.lu

PRACE SoHPCSoftware applied

ANSA, OpenFOAM, ParaView

PRACE SoHPCMore Information

Paolo Scuderi
Benet Eiximeno Franch

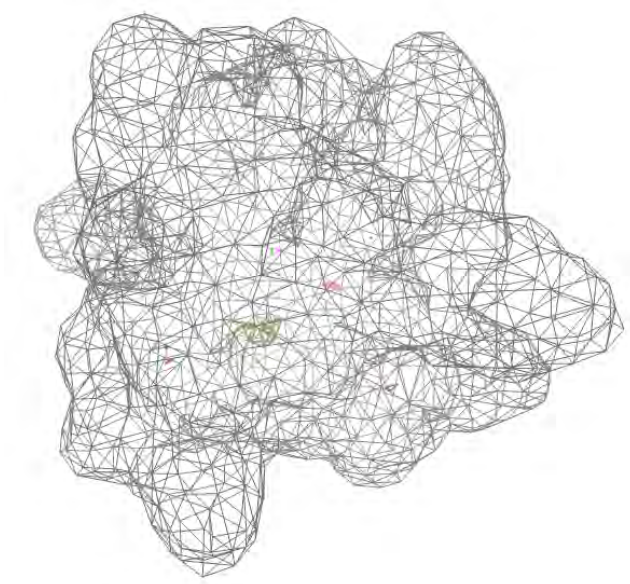
PRACE SoHPCAcknowledgement

Dr. E. Krishnasamy for mentoring. Dr. S. Varrette and Dr. A. Olloh for the technical support. Dr. Leon Kos and the PRACE SoHPC 2021 staff, for the great opportunity in Europe with HPC.

PRACE SoHPCProject ID

2131

HPC and Molecular Surfaces



Ulaş Mezin, Miriam Beddig

The aim of the project is to extend and improve the implementation of the Marching Tetrahedra algorithm. This algorithm is currently used to determine biomolecular surfaces based on an approximation of the electron density.

This project deals with the computation of molecular surfaces. The goal is to continue the efforts already initiated in last year's SoHPC project 2024, where a novel approach based on Marching Tetrahedra had been developed from scratch.

The shape of a molecule can be defined in various ways.[1] In this project, we are focusing on modelling the molecular surface as an isosurface of the so-called *electron density*. The electron density represents the probability of finding an electron in a specific location around the molecule. Since it decreases exponentially with the distance to atomic centres, a small enough chosen threshold value can serve to mark off the overall shape of the molecule under investigation.

Several algorithms are available to compute isosurfaces. One approach is the *Marching Tetrahedra*[2] algorithm which has been taken into account in this project. It is possible that the resulting isosurface is not a single closed compound but rather consists of several closed surfaces (see Fig 1). However, for virtually all physical applications that build on molecular surfaces it is necessary to be able to differentiate algorithmically between different closed surfaces (red or blue in Fig 1).

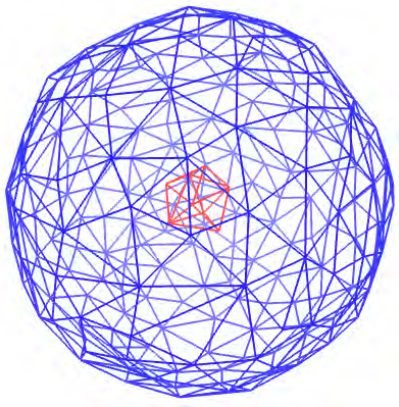


Figure 1: Surface of Buckminster Fullerene, C60. The surface is a composition of an inner surface (red) and an outer surface (blue).

A very general approach — considered last year — is the computation of the Euler characteristic.[3] The Euler characteristic can be used to determine if the resulting isosurface consists of a single layer or if it is a composition of multiple closed surfaces. However, this approach is computationally demanding and it is not possible to distinguish different surfaces algorithmically. Consequently, one goal of this year's project is to tackle this problem.

Since the program shall also work for large molecules, one further goal of this year was the improvement of the efficiency of the computations.

Distinguishing different surfaces

A literature review was performed to find possible algorithms which tackle the mentioned problem of distinguishing different surfaces.

During the review, the *Voxel Sweep* algorithm was found.[4] This algorithm is an extension of the Marching Cubes algorithm and is supposed to solve the open problem outlined above. To begin with, independently of the existing code, a 2D version of the algorithm was implemented. This was done with Python3 and the Jupyter Notebook (see Fig 2).

The implementation of the 2D Voxel Sweep algorithm runs in strictly sequential order over the grid. We explored

how the algorithm can be parallelized.

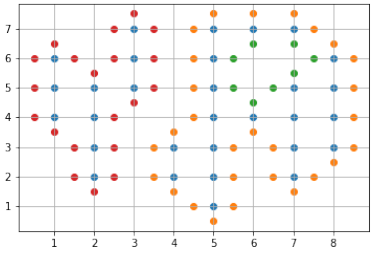


Figure 2: Result of 2D Voxel Sweep: The blue dots represent grid points in the interior of some object. The dots in red, green and yellow describe resulting surfaces. The algorithm correctly detects three independent surfaces and colours them differently.

For that, the grid was divided into sub domains which don't overlap. Then, the Voxel Sweep algorithm was applied on each sub domain independently (see Fig 3). After that, an additional merge was carried out that finalised the open space between individual sub domains.

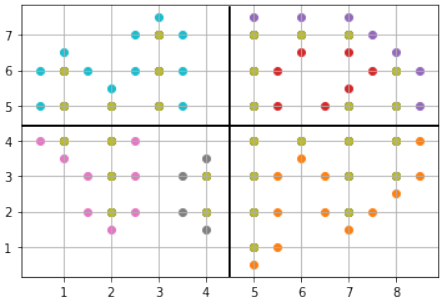


Figure 3: The Voxel Sweep algorithm is applied to each sub domain independently. The remaining step is to merge the surface components of each subgrid.

We implemented the 3D Voxel Sweep algorithm in C and inserted it into the existing program. The code creates for each disconnected surface a file in PDB format (protein data bank). With VMD (molecular visualization program), the results can be graphically displayed and checked for correctness.

To check if the Voxel Sweep algorithm is still feasible for large input molecules, the code was examined with the profilers gprof and ARM/FORGE. From the profiles, some I/O overhead could be identified. We adapted the code and substituted it with something more efficient. After doing that, we could not find any major performance impacts by the Voxel Sweep algorithm.

For applications it is desirable if very small surface components can be identified and sorted out automatically, since these surfaces don't have a physical relevance. In order to achieve this, we computed the volume and surface area of every single surface component. Then, if the volume or area of a surface falls under the limit of a certain threshold value, the surface can be sorted out automatically.

The computation of the surface area and volume of a molecule was already implemented in the existing code. However, it was only a rough approximation. The distinction between different surfaces made it possible to improve these computations.

Performance Tuning

Each of the atoms needs to compute its contribution to the electron density at various locations on the grid. A schematic representation of the workflow of last years implementation is given in Fig 4.

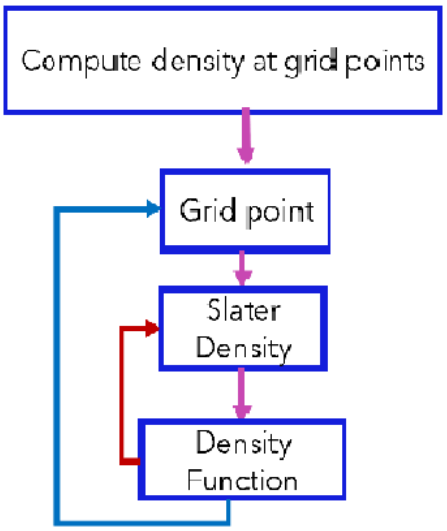


Figure 4: Workflow of function *compute_density_at_grid_point()*

All operations are carried out in function *compute_density_at_grid_point()*. Since we are dealing with a 3-dimensional grid, the number of operations will grow with the third power of the dimension of the grid (i.e. number of grid points considered along a single axis). After specifying a particular grid point, two device functions (to be executed on the GPU only) are called in sequence i) *slater_density()* and ii) *densityH/C/O...()*, where the former is a kind of selector for the latter functions which each handle a particular type of chemical element. Being the key component in terms of computational cost, optimization studies were carried out to identify optimal conditions for making efficient use of modern GPU architectures.

Optimization Details

In the course of optimization, 11 different code versions were prepared and analyzed. Here version 5 refers to last years implementation. All individual optimization steps showed continuous progress towards improved runtime conditions. Details are outlined below.

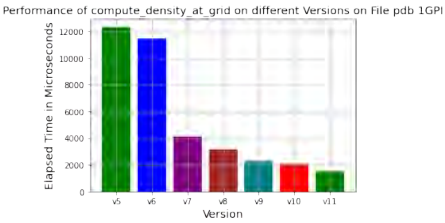


Figure 5: Performance improvements of different code versions for pdb structure 1GPI

Version 6

In this version the device function pointer is used to eliminate if-statements. For this purpose, when the PDB file is read, the atoms are indexed from largest to smallest according to the ASCII table. This made the code much more readable. After it was profiled, memory overhead was observed. The achieved performance improvement was on the order of 10% (see Fig 5, blue versus green bar). However, given the fundamental change in code structure, version 6 was not considered further for additional optimization steps.

Version 7

Slater densities are calculated according to different atom types. Various parameters and constants are being used following the physical definition of orbitals corresponding to individual chemical elements. Here these parameters are pre-calculated and assigned fixed values instead of being re-calculated over and over again when calling different density functions. A significant speedup of approximately 3x could be achieved from such type of processing (see purple bar in Fig 5).

Version 8

Density computations include many exponential operations and distance calculations. These processes are very costly and may be better performed from specially tuned libraries. For this reason substitution with Fast-Math calls was

examined next. It was observed that results did not change for a test set of 7 different PDB files when switching to Fast-Math calls. An additional 33% performance gain could be achieved (see dark-red bar in Fig 5).

Version 9

Profiling revealed that the number of instructions for creating and deleting local variables in each of the device functions made up the majority of issued instructions. Thus, following the principle of minimizing function calls, all functions were gathered under a single parent function — *compute_density_at_grid_point()*. Moreover, repeated access to atoms' coordinates (residing in global memory) was replaced with a single copy into local memory and several of the re-iterated parameter assignments were converted into static type of numerical constants. These changes finally led to an improvement of approximately 50% over the previous version (compare cyan to dark-red bar in Fig 5).

Version 10

Natural occurrence of different atom types was estimated from analyzing a couple of pdb files. Based on these findings the order of querying different atom types was revised so that the most frequent type (H-atoms) was considered first and further types followed according to frequency of occurrence. This modification provided an additional 20% performance gain (see red bar in Fig 5).

Version 11

All data types were converted from double to float as 7 digits behind the decimal point were considered sufficient quality of numerical accuracy. This way memory-bound and compute-bound limits are implicitly optimized for different types of GPUs and the time spent in memory allocation is reduced as well (about 30%, see Fig 6 and dark-green bar in Fig 5).

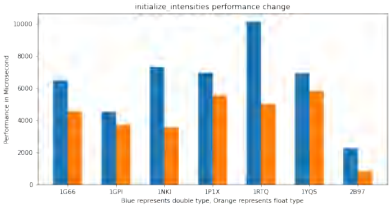


Figure 6: Performance gains from data type conversion as noticed in memory allocations

Results

The iterative optimization process has led to a final code version (v11) in which all computational operations were aggregated in function *compute_density_at_grid_point()*. Here, each thread performs operations in a 3-fold nested loop on a local subsection of the grid. The obtained results show between 8x acceleration for larger sized molecules and 13x acceleration for smaller sized molecular structures (see Figs 7-8). Significant improvements could be made from converting data types from double to float both in terms of times spent in memory allocation as well as relative dedication to compute-bound versus memory-bound fractions of the algorithm for all various types of GPU architectures. Further benefits were achieved with respect to a reduction in energy consumption and a shortening of compilation times.

Version/File	1G66	1GPI	1NKI	1P1X	1RTQ	1YQ5	2B97
Version 5	8885.5	12329.0	7046.4	8992.2	9754.0	9487.1	6669.9
Version 6	8343.0	11493.0	6863.9	8705.7	8621.0	8497.3	6577.9
Version 7	3966.3	4169.6	2465.4	3598.6	3603.3	4456.1	2056.5
Version 8	1955.0	3192.9	1588.2	2062.3	3175.7	2475.8	1137.4
Version 9	1377.2	2316.9	867.2	2051.7	2731.0	2267.5	604.64
Version 10	1357.9	2087.9	983.68	1879.4	2436.3	2453.8	602.15
Version 11	1067.6	1559.6	657.35	1429.3	1722.0	1603.7	431.46

Figure 7: Execution times of different code versions applied to 7 different pdb structures.

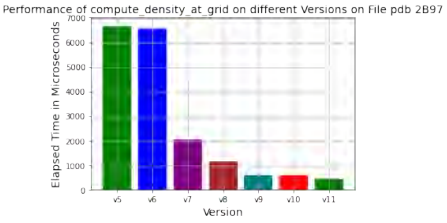


Figure 8: Performance comparison of different code versions for pdb structure 2B97

Discussion

Even for individual threads involved in a sequence of loops it is possible to perform operations efficiently based on a maximized set of local data. Following our results, the most suitable place for the storage of fixed data is the register. From examining PDB files a rough frequency of occurrence of different atom types can be derived as H: 49%, C: 32% O: 11%, N: 9%, S: 1%. Taken this into account in the context of SIMD, we can hypothesize the active warp number as a maximum of 49%. However, memory spaces available to each of the atoms could be shared memory, constant memory or texture memory. Since we would access the same locations in shared memory, parallel operation is hampered. On the other hand, even if we assume a broadcast structure in constant memory, access to the same location was proportional to the relative frequency of different atom types. The texture engine would provide better performance especially in the spatial locality of 2D and 3D data. For this reason, we did not use these memory spaces as we could not benefit from any spatial locality in the random access scenarios anticipated here. To this end we preserved the for-loops and kept the values to be accessed in the register. The number of data to be kept in the register is the same as the number of shells for each atom. Therefore, according to version 5, a maximum of 3 extra registers are allocated. Thus, maximum performance can be achieved by providing minimum occupancy.

References

[1] Gui, S., Khan, D., Wang, Q., Yan, D. M., and Lu, B. Z. (2018). Frontiers in biomolecular mesh generation and molecular visualization systems. Visual Computing for Industry, Biomedicine, and Art, 1(1), 1-13.

[2] Chan, S. L., and Purisima, E. O. (1998). A new tetrahedral tesselation scheme for isosurface generation. Computers and Graphics, 22(1), 83-90

[3] https://en.wikipedia.org/wiki/Euler_characteristic

[4] Cohen, I., and Gordon, D. (2004, December). The Voxel-Sweep: A Boundary-based Algorithm for Object Segmentation and Connected-Components Detection. In VMV (pp. 405-411).

PRACE SoHPCProject Title

HPC Implementation of Molecular Surfaces

PRACE SoHPCSite

VSC Research Center, Austria

PRACE SoHPCAuthors

Ulaş Mezin, Miriam Beddig,

PRACE SoHPCContact

Ulas Mezin
Izmir Institute of Technology
E-mail: ulasmezin@gmail.com
Miriam Beddig
University of Edinburgh
E-mail: s2110240@sms.ed.ac.uk

PRACE SoHPCMentor

Siegfried Hoefinger, Vienna, Austria

PRACE SoHPCCo-Mentors

Markus Hickel, Balazs Lengyel, David Fischak, Vienna, Austria

PRACE SoHPCAcknowledgement

We would like to express our gratitude to our mentor Siegfried Hoefinger for providing us the guidance throughout the project. Furthermore, We would also like to thank PRACE for giving this opportunity to work at VSC research centre, Vienna.

PRACE SoHPCProject ID

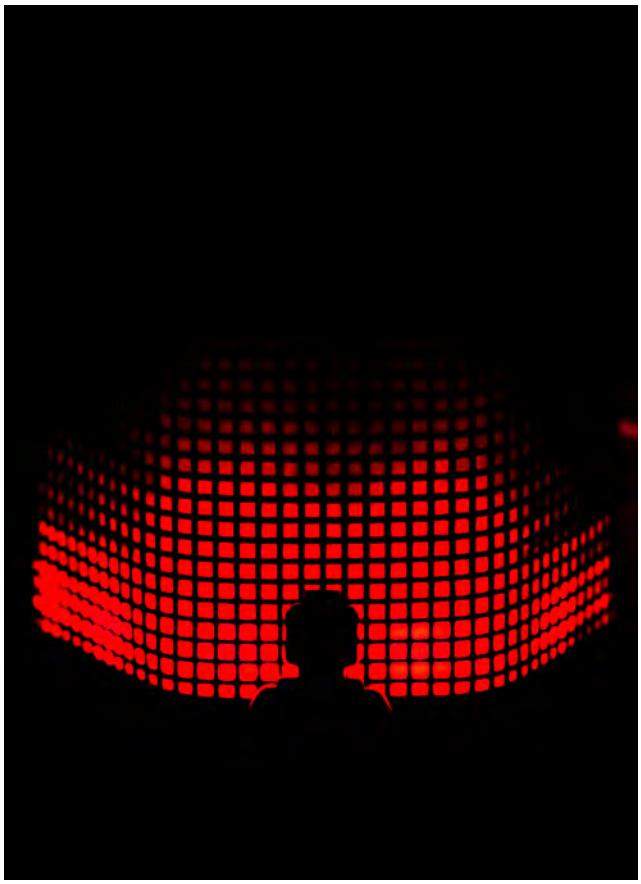
2132

Miriam Beddig

Ulaş Mezin

Big Data meets HPC

Rajani Kumar Pradhan, Pedro Hernandez Gelado



Abstract Scientific research can benefit laterally from Big Data tools developed in the last decades for commercial data management, such as Hadoop and Apache Spark, by leveraging their use with traditional High Performance Computing resources. High degrees of automated parallelism can be trivially achieved to optimize read, write and simple transformations. Nevertheless, certain complex operations are still challenging to perform within Spark, traditional HPC can still fill these gaps.

1 Introduction

It is estimated that every single day 2.5 quintillion bytes of data is produced in the whole world. How much of it can we store? And how much of it can we analyze? Data is becoming increasingly commoditised and the tool kits developed to manage it offer huge potential, however they still lack a wide adoption in an academic context for scientific research in non-computer science fields.

Apache Spark has become the "state-of-the-art" framework in Big Data and it offers huge untapped potential to academic researchers that deal with challenging to process amounts of data in conventional systems. Scientific problems are now not only processor-bound tasks, they are increasingly becoming memory-bound problems. In this context we studied two prototypical scientific applications that could benefit from applying Big Data techniques coupled with traditional HPC (High Performance Computing). Our objective: to optimize workflows, leverage cluster resources, increase parallelism and use distributed data storage.

The first use case will deal with analyzing the Integrated Multi-Satellite Retrievals for GPM (IMERG), a satellite precipitation product from the Global Precipitation Measurement (GPM) mission by the National Aeronautics and Space Administration (NASA) and Japan Aerospace Exploration Agency (JAXA). The IMERG precipitation datasets are at a daily scale and in NetCDF format, covering from 2000 to 2020 which has around 230 GB in total. The processing and analysis of these datasets is a real challenge ahead of the scientific community, especially for R users. In this context, the main objective of this case study is to explore the state-of-the-art of Spark architecture and framework to analyze and process the geospatial (e.g., IMERG) datasets in a more efficient way.

Our second use case deals with another area of research that is becoming more and more memory-bound: post-processing Computational Fluid Dynamics (CFD) simulations. Aerodynamic researchers are dealing with larger file outputs due to several reasons, amongst them: the improvement in mesh refinement, more complex geometries, and unsteady aerodynamic research, that is through many evolving time-steps. Open-source CFD tools like Open-Foam and Paraview are able to leverage resources in the researchers machine, but how can we optimize CFD post-

processing codes to use clusters? This second use-case will analyze the opportunities, challenges and pit-falls of using Apache Spark in this context.

2 Methods

In developing this project, we have designed an approach for budding Spark users who are attempting to optimize and port their data-intensive tasks to Spark for the first time. We identified some crucial steps involved with porting an application to an Hadoop environment and evaluated performance improvements as well as implementation challenges in a hands-on approach. We hope that our process will serve as a blueprint for similar endeavours. Benchmarks on each of these steps will be developed in the results section of this report. However, a rough heuristic to calculate the ratio of time-savings and performance improvements to input work from the researcher when applying these steps can be summarized as 1,2,3,4. The easiest time savings can be obtained in steps 1 and 2, whilst steps 3 and 4 require more work from the user.

2.1 Step 1: Read into Spark

Our strategy's first step is to read, preprocess and transform our serial scientific application's data, for example a .csv file, into a Spark data abstraction like RDDs, a Dataframe or parquet files.

2.2 Step 2: Apply pre-built Spark functions

If you can, always substitute any functions from the serial code you are porting with native Spark functions. Trivial functions that can be performed with simple SQL queries, groupBy, filters or map-reduce operations work best here.

2.3 Step 3: Optimize loops

More complex fors, whiles, recursiveness, loops are the next bottlenecks we need to tackle. How can we port them to Spark if we can't find any native Spark substitutes? Use and exploit user-defined functions were you can perform row-wise operations using lambda functions. At all costs, avoid collect() and other actions that extract and serialize RDD data from Spark, slowing you down!

2.4 Step 4: Test small, build big

Always begin testing with very small excerpts from your large dataset. Jupyter Notebooks is a fantastic tool to develop your code as you can test these small samples and benchmark early on against your legacy serial version. HINT: use the '*%%time*' function in Jupyter to time cell execution. At any point, if you run into trouble, try to create minimum reproducible examples to share, many online forums including Stack Overflow have vibrant communities in which you can share Spark issues and questions. Once you are ready to build, create your slurm files and submit to your HPC system.

3 Results

Results are shown here for the two prototypical use cases studied.

3.1 Reading and processing of IMERG datasets

The main bottleneck of dealing with NetCDF datasets is that the data is stored in multiple dimensions. Therefore, before doing any analysis the detests should be stored in a more user-friendly format such as an R data-frame or csv (I prefer into an .RDS format). The R script to convert Netcdf to an R data-frame was slow, which takes approximately 220 seconds for 30 files each with 30MB on the Little Big Data Cluster (LBD) which has 18 nodes each with 48 cores. That is, 864 virtual cores and 4.5TB of RAM. However, as the R script is designed for the single machine it does not really use the benefits of multi-core clusters. Therefore, the same R script was applied through the spark.lapply() function, which parallelize and distribute the computations among the nodes and their cores.

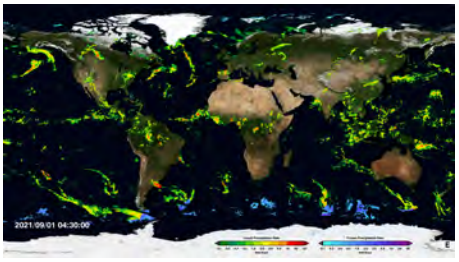


Figure 1: IMERG image from NASA more info on the dataset can be found here.

The bench-mark (run for 5 times) results show that (Table 1), comparatively SparkR has significantly faster than R. For instance, on average, R takes 220 seconds to complete the process, whereas for SparkR it was just 26 seconds, which is approximately 8 times as much as faster than R.

Table 1: Benchmarking R versus SparkR in reading and extracting one month of IMERG daily datasets (units are in seconds).

Method	Minimum	Mean
R	225	220
SparkR	25	26

Furthermore, the benchmarking is also performed for one month of datasets (30 files each with 30 MB) and with three different write functions i.e., .CSV, .RDS and Parquet formats. Results find that extracting the variable from each datasets and writing the output into a paraquat format is much faster than writing into a .RDS or into CSV format. Additionally, reading the paraquat file is relatively faster than reading a .RDS format and CSV.

3.2 CFD Post-processing

In this use case a Gamma-1 method code in Python that can be found here was ported into Spark with variable success. The Gamma-1 method is computed using the following formula:

$$\Gamma_1(P) = \frac{1}{N} \sum_S \frac{(\mathbf{PM} \times \mathbf{U_M}) \cdot \mathbf{z}}{||\mathbf{PM}|| \cdot ||\mathbf{U_M}||} \quad (1)$$

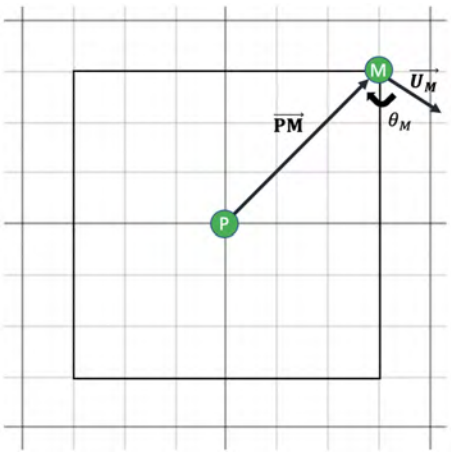


Figure 2: Gamma-1 method, as visible requires a calculation of the nearest neighbors before computing the vector multiplication.

3.2.1 Read Improvements

By using Spark we can obtain significant performance upgrades from our serial code in Pandas were it took a mean μ= 22 minutes and 19 seconds to perform a read operation of 18 million records, about 500mb of data, using a for loop. Instead, in Apache Spark we were able to parallelize this read to leverage 72 virtual cores, optimizing it to a mean read time of μ= 41.3s, a 3240% increase in performance. Tests were performed five times to account for variable cluster loads and capacity, standard deviation is cited as σ.

Table 2: Read times measurements for 5 samples, mean μ and standard deviation σ

Method	μ [s]	σ [s]
Serial for loop	1339	107
Spark Dataframe	41.3s	4.0

3.2.2 Neighbours in radius

The author attempted to optimize a Euclidean radius neighbours search with Spark using a native locality-sensitive hashing (LSH) implementation in the Apache Spark library MLlib known as BucketedRandomProjectionLSH. However poor performance improvements, in fact degraded performance, was obtained. This, even when accounting the fact that the legacy function performed an exhaustive itemized serial search of 47,000 records for each run. Instead the LSH model built once for every query in its vanilla implementation, and an enhanced LSH precomputed model (only built once) repeatedly queried for each point. The test results were obtained from five runs and cited on a per point queried for neighbours basis, the Spark implementations used 400 cores vs one serial core.

Table 3: Radius neighbours search performance

Method	μ [s]	σ [s]
LSH vanilla	6.58	0.63
LSH precomputed	4.87	0.75
Legacy	0.484	0.0358

3.2.3 Gamma-1

Gamma-1 computation for a record within Spark requires a cascading set of UDFs(User-defined functions) to be performed, first a radius nearest neighbours search for each point, followed by the computation of the gamma-1 method for it.

Unfortunately, this is not possible in Spark, as worker nodes/executors (already performing an action) cannot call a further action upon other executor nodes. This is part of the basis of the Spark architecture and a key barrier in creating automatic parallelism for scientific applications that have cascading algorithms, were the results of one operation are fed and called by another function recursively. The author hence struggled to completely port the code, even after porting all the separate components necessary to complete it.

3.2.4 Stagnation Point Location

Finally another common CFD post-processing operation in unsteady fluid mechanics, when searching for areas with high probability of a vortex core location, is to locate stagnation points in the flow, that is points where the velocity is approximately 0. Spark shines in this type of single filtering and retrieving operation in comparison with locating these serially in pandas or even using OpenFoam. The test input files were used, that is 400 frames with 47,000 records each or approximately 1.8 million records.

Table 4: Stagnation points

Method	μ [s]	σ [s]
Vanilla pandas	143	1.96
Spark	21.41	3.4
OpenFoam	2-5s	N/A

4 Discussion & Conclusions

Spark has been shown as extremely powerful tool in the management, wrangling and native function analysis of large amounts of scientific data. In this, it offers significant performance improvements in comparison with serial Python read, write and simple transformation codes, whilst leveraging cluster and HPC-level parallelism trivially. However, more complex operations that do not have native Spark equivalents in official or supported libraries, specially those in which cascading algorithms are computed repeatedly, are beyond the scope of what Spark is ideally suited to perform easily, and are challenging for the researcher to port from Python into the Apache Spark framework.

In the specific case of CFD post-processing Spark, offers large potential to a researcher in the management and simple analysis of their CFD data, especially for unsteady flow problems with fine meshes that are difficult to post-process serially. Nonetheless, it is not trivial to perform complex operations on this data in Spark, which shines in filtering, aggregation, and native machine learning library tools. This still leaves room for further work into looking how to transfer certain workloads from Spark to mpi, and vice-versa, to be able to have the "best of both worlds".

Future work could study the possibility of using new Spark 3.12 UDAFs (User defined aggregated functions), Koalas and other tools that allow easy-porting of python code into Spark, as well as stragies on how to integrate classical HPC parallelism into hybrid Spark-HPC applications. Into this feeds studying how to best integrate proven highly scalable parallelism frameworks like mpi to Big Data tools like Spark to best aid the researcher attempting to deal with large files in their scientific research.

5 Acknowledgments

We want to profusely thank Giovanna Roda, Dieter Kvasnicka, Claudia Blaas-Schenner, Liana Akobian and Vienna Scientific Cluster (VSC) for hosting us during the summer. The team feeling, experience and cheery atmosphere has been fantastic at VSC, and we have learned lots thanks to them. Moreover we want to thank Leon Kos and PRACE for offering us the opportunity to work in this fantastic subject. Thank you.

PRACE SoHPCProject Title
The convergence of HPC and Big Data/HPDA

PRACE SoHPCSite
Vienna Scientific Cluster, Austria

PRACE SoHPCAuthors
Rajani Kumar Pradhan, Pedro Hernandez Gelado

PRACE SoHPCMentor
Dr. Giovanna Roda, Technical University of Vienna, Austria

PRACE SoHPCContact
Leon, Kos, Univerza v Ljubljani
Phone: +12 324 4445 5556
E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied
Hadoop, Python, Apache Spark

PRACE SoHPCMore Information
spark.apache.org hadoop.apache.org python.org

PRACE SoHPCProject ID
2133


Rajani Kumar Pradhan


Pedro H. Gelado

www.summerofhpc.prace-ri.eu