PRACE

PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

2019

summerofhpc.prace-ri.eu

A long hot summer is time for a break, right? Not necessarily! PRACE Summer of HPC 2019 reports by participants are here.

HPC in the summer?

Leon Kos

There is no such thing as lazy summer. At least not for the 25 participants and their mentors at 11 PRACE HPC sites.

Summer of HPC is a PRACE programme that offers university students the opportunity to spend two months in the summer at HPC centres across Europe. The students work using HPC resources on projects that are related to PRACE work with the goal to produce a visualisation or a video.

This year, training week was in Bologna and it seems to have been the best training week yet! It was a great start to Summer of HPC and set us up to have an amazing summer!

At the end of the summer videos were created and are available on Youtube as PRACE Summer of HPC 2019 presentations playlist. Together with the following articles interesting code and results are available. Dozens of blog posts were created as well. At the end of the activity, every year two projects out of the 25 participants are selected and awarded for their outstanding performance. Award ceremony was held early in November 2019 at PRACE AISBL office in Brussels. The winners of this year are Mahmoud Elbattah for Best Performance and Arnau Miro Janea as HPC Ambassador. Benjamin surpassed expectations while working on an interesting project, and carrying out more work than planned. Mostly with no assistance, he carried out benchmarking analysis to compare different programming frameworks. His report was well written in a clear and scientific style. Pablo carried out a great amount of work during his project. More importantly, he has the capacity to clearly present his project to laypersons and the general public. His blog posts were interesting, and his video was professionally created and presented his summer project in a captivating and pleasant way.

Therefore, I invite you to look at the articles and visit the web pages for details and experience the fun we had this year.

What can I say at the end of this wonderful summer. Really, autumn will be wonderful too. Don't forget to smile!

Contents

PRACE SoHPC2019 Coordinator
Leon Kos, University of Ljubljana
Phone: +386 4771 436 E-mail: leon.kos@lecad.fs.uni-lj.si

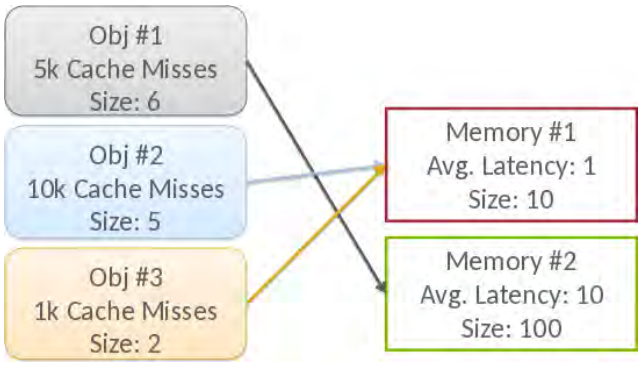
PRACE SoHPCMore Information
http://summerofhpc.prace-ri.eu


Leon Kos

Memory: Speed Me Up

Dimitrios Voulgaris

Memory interaction is one of the most performance limiting factors in modern systems. It would be of interest to determine how specific applications are affected, but also to explore whether combining diverse memory architectures can alleviate the problem.



Supercomputers are gradually establishing their position in almost every scientific field. Huge amount of data requires storage and processing and therefore impels the exascale era to arrive even sooner than anticipated. This immense shift cannot be achieved without seeing the big image. Processing power is usually considered as the determinant factor when it comes to computer performance, and indeed, so far, all efforts in that direction did seem fruitful. CPU performance, however, has been extensively researched and thus optimised, leaving scarcely any opportunities for improvement.

Introduction

Heterogeneous memory (HM) systems accommodate memories featuring different characteristics such as capacity, bandwidth, latency, energy consumption or volatility. While HM systems present opportunities in different fields, the efficient usage of such systems requires prior application knowledge because developers need to determine which data objects to place in which of the available memory tiers [1]. Given the limited nature of "fast" memory, it becomes clear that the approach of placing the most often accessed data objects on the fastest memory can be quite misleading. In order to identify the data which benefit the most from being hosted into different memory subsystems it is important to gain insight of the application behaviour.

Application profiling comes in two flavors, software and hardware based. Tools like EVOP [2], which is an extension of Valgrind [3], VTune [4] as well as others, offer exclusively instruction-based instrumentation, i.e. monitoring and intercepting of every executed instruction. The additional time overhead implied by this technique is more than apparent, therefore, vendors have come up with the idea of enhancing hardware, making it capable of aiding in application profiling. Specialized embedded hardware counters deploy sampling mechanisms to provide rich insight in hardware events. PEBS [5] is the implementation of such mechanism in the majority of modern processors.

These two approaches essentially have the same ultimate scope of highlighting the application behaviour regarding its memory interaction in order to result in an optimal performance-wise memory object distribution. Nevertheless the former approach accounts for each and every memory access while the second one performs a sampling of the triggered events. It would be of great interest to find out whether by forcing software to imitate hardware's methodology, i.e. sampling, we are capable of achieving similar final results. This the original target of the project.

Background

Memory architecture, describes the methods used to implement computer data storage in the best possible manner. "Best" describes a combination of the fastest, most reliable, most durable, and least expensive way to store and retrieve information [6]. Traditionally and from a high point of view, a memory system is a mere cache and a RAM memory.

Caches are very specialised and hence complicated pieces of hardware, placed in a hierarchical way very close to the processing unit. Divided into layers (usually 3 in modern machines), every cache element presents different characteristics regarding size and data retrieval latency.

RAM, being an equally complex piece of hardware, presents significantly bigger capacity which qualifies it as the ideal main storage unit of the system. On the downside, it is characterised by degrees of magnitude longer access time which makes it quite inefficient, yet necessary to access.

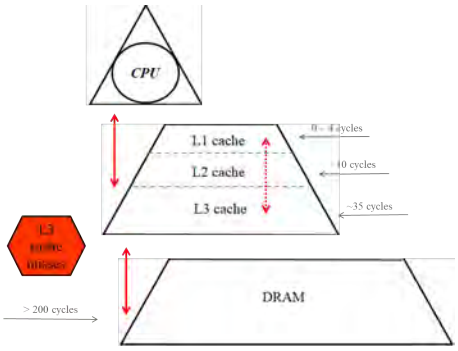


Figure 1: Memory hierarchy along with typical latency times.

Figure 2: Valgrind logo.

```
simulate:
  CALLGRIND_START_INSTRUMENTATION;
  CALLGRIND_TOGGLE_COLLECT;
     /* Original code simulation */
  CALLGRIND_TOGGLE_COLLECT;
  CALLGRIND_STOP_INSTRUMENTATION;
```

Figure 3: Pseudo-code for enabling Valgrind instrumentation and detail collection.

Both storage elements interact with each other in the following way: when a memory access instruction is issued, cache memory is the first subsystem to be accessed. If the respective instruction can be completed in cache (that is: the referred value resides there) then we have a *"cache hit"*, otherwise a *"cache miss"* will be signaled and the effort of completing the instruction shall move on to the next cache level. In case of a *"last level cache miss"* the main memory has to be accessed in order to fulfil the need for data. These exact accesses pose the greatest performance limitation especially when it comes to memory-bound applications. Keeping that in mind, these exact accesses have we considered as the key factor for optimising our workflow.

Methodology

In order to monitor memory interactions we opted for Valgrind as it offers the additional possibility of simulating cache behaviour. That is, when we run our application under Valgrind, all data accesses can be monitored and collected giving the chance of determining all those accesses that missed cache and had to deploy memory subsystems deeper in the hierarchy.

Considering every accessed data in isolation would be neither intuitive nor helpful for further post-processing. This is where EVOP intervenes in order to group data into memory objects. An object can be an array of integers, a struct or any other structure the semantics of which suggest that it has to be considered as an entity. Such objects, regardless if they are statically or dynamically allocated are labeled and every time an access is issued, it is tied to the related object.

Simulating, intercepting and keeping logistics of that a few billion data (memory references) can be highly computationally intense. Indeed, instrumenting an application will make it from four to twenty times slower. In order to alleviate this fact we decided to restrict the instrumentation only to the regions of code that is of interest. Valgrind provides the adequate interface that eases the aforementioned scope: In figure 3 you can see the macros that, by framing the relevant lines of code, enable or disable the instrumentation and information collection.

Setting the aforementioned comparison of software and hardware approach as our ultimate scope, we enhanced EVOP with the option of performing sampled data collection. In detail, we extended Valgrind source code by integrating a global counter which increments on every memory access. While counter's value is below a predetermined threshold no access shall be accounted for. On the contrary, the memory access that forces counter and threshold to be equal will be monitored and therefore accumulated to the final access number. The threshold is to be plainly defined when EVOP is called using the flag `--sampling-period=<int>`.

What is more, we familiarised with the internal Valgrind representation of memory accesses in order to distinguish between *"load"* and *"store"* ones. Similarly as before, we added a counter that increments every time a load or store is detected. The sole accesses that are to be monitored are the ones that equalise the counter to the predetermined threshold. This time, the user has to define a combination of flags such as `-sample-loads=no|yes -sample-stores=no|yes -sampling-period=<int>` in order to specify the type of accesses to be sampled as well as the respective sampling period.

Having enabled these two features our methodology can be summarised in the following procedure: We initially performed a simulation of our benchmarks without sampling in order to get the total number of memory objects. The latter were processed by a BSC-developed tool in order to be optimally distributed to the available memory subsystems. The distribution takes into consideration the last-level cache misses of each object as well as the number of loads that refer to each one of them and sets as its goal the minimisation of the total CPU stall cycles. The total "saved" cycles are calculated along with the object distribution and thus the final speedup can be determined. This speedup is the maximum achievable given the application and the memory subsystem mosaic.

What follows is a trial-and-error experimentation process of obtaining results using various sampling periods to extract the referenced objects. It can be intuitively assumed that the longer the sampling period is, the fewer the total memory accesses will be, too. Given the fact that each access is related to one memory object, the fewer the accesses are, the less the possibilities are that not all existing memory objects are discovered. The latter presumably results in a worse distribution; worse in the sense that the final speedup is lower than the initial, achievable one. Nevertheless, depending on the access pattern of each benchmark, there is a specific sampling period, or a restricted range of neighbouring sampling periods, that identify all the objects responsible for the initial speedup.

Note that potential access patterns may exist among code loops. A sampling period that always intercepts the same memory access (in different loop) is considered faulty and shall result in biased outcome. In order to avoid that, we used exclusively prime numbers as sampling periods.

Test Cases – Results

Since memory behaviour is of essence for this project it is important to choose wisely the applications-under-test. MiniMD and HPCCG are a pair of

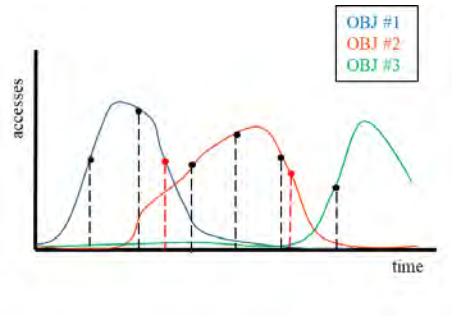


Figure 4: Sampling at a specific low rate results in omitting memory objects; a higher sampling frequency is needed

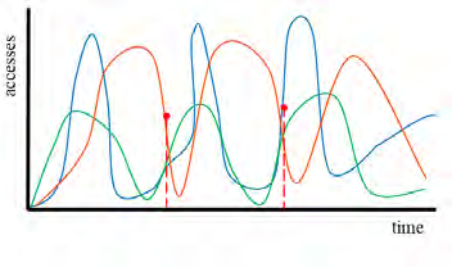


Figure 5: In rather iterative access behaviour, low-frequency sampling is enough for complete memory object discovery.

mini-applications intended as benchmarks for assessing the performance of certain production applications. Hence, they are representative of the behaviour of more complex applications.

MiniMD, a molecular dynamics simulator, has a rather sequential access pattern of the objects that it refers to. As a result, sampling period cannot be of the same degree of magnitude of the total access number (discovered by the non-sampled execution). In fact, in order to have an optimal speedup, sampling period has to be relatively small. On the contrary, HPCCG, which offers an iterative computational implementation of the conjugate gradient method on an arbitrary number of processors, entails its main computation in a "for loop". Thus, the same memory objects are referenced every time the loop iterates enabling the use of greater sampling periods. Indeed, we were able to use intervals of the same degree of magnitude as the overall access number of some objects and still get accepted speedup time.

In the first two figures we present a qualitative representation of the memory access behaviour for each application, as it was understood by us. Sparse sampling, in the first case, is not enough to account for every "important" object,

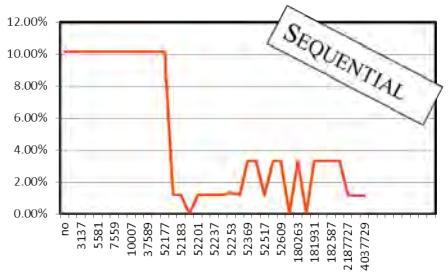


Figure 6: Only low periods are allowed in order to achieve an optimal speedup.

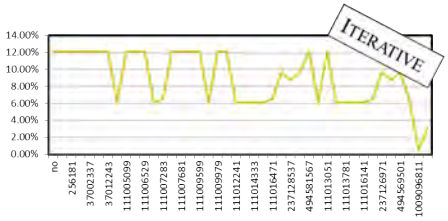


Figure 7: Bigger sampling periods can result in equally high speedup as memory objects keep being discovered.

while in the second case, due to the different access pattern, this is allowed. In the following two graphs we pictured the exact correlation between the final speedup and the sampling period used to get the objects. While in the first case, there is a strict threshold, after which the final results are strongly damaged, in the second case this threshold can be generalised in a larger neighbourhood of periods.

Future Research

When deploying the hardware profiling approach aiming in performance optimisation we are restricted by the imposed sampling that has to be performed. So far we have obtained some concrete results regarding the effect that different sampling rates have on the discovered memory objects. Given that the latter are responsible for the final performance speedup, as well as that hardware profiling mechanisms rely solely on sampling, we can trivially assume that the same, or at least similar, results should be obtained when profiling using the hardware counters.

We have provided the chance to future researchers (and why not future SoHPC participants) to experiment using the sampling periods from our trials in order to determine the most efficient way to gain insight into an application's memory behaviour and subsequently boost its performance by exclusively focusing on the optimal distribution of its memory objects.

References

[1] H.Servat, A. J. Peña, G. Llort, E. Mercadal, H. C. Hoppe, J. Labarta, "Automating the Application Data Placement in Hybrid Memory Systems", 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, September 5-8, 2017.

[2] A. J. Peña, P. Balaji, "A Framework for Tracking Memory Accesses in Scientific Applications", 2014 43rd International Conference on Parallel Processing Workshops, Minneapolis, MN, USA, September 9-12, 2014.

[3] "Only Intel® VTune™ Amplifier", Available: https://software.intel.com/en-us/vtune

[4] "Valgrind", last version: valgrind-3.15.0, Available: http://www.valgrind.org/

[5] I. Corporation, Intel 64 and IA-32 Architectures Software Developer's Manual, September 2016, vol. Volume 3B: System Programming Guide, Part 2, ch. 18.4.4.

[6] https://en.wikipedia.org/wiki/ Memory_architecture

[7] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C.Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist,and R. W. Numrich, "Improving performance via mini-applications,"Sandia National Laboratories, Tech. Rep., 2009, http://www.sandia.gov/ maherou/docs/MantevoOverview.pdf

PRACE SoHPCTitle

Analysing effects of profiling in heterogeneous memory data placement

PRACE SoHPCSite

Barcelona Supercomputing Center (BSC), Spain (ES)

PRACE SoHPCAuthors

Dimitrios Voulgaris [University of Thessaly] — Greece (GR)

PRACE SoHPCMentor

Marc Jorda [BSC] — Spain (ES)

Dimitrios Voulgaris Photo

PRACE SoHPCContact

Dimitrios, Voulgaris, University of Thessaly
E-mail: jim.voulgaris@gmail.com

PRACE SoHPCAcknowledgement

Special thanks to my BSC coordinators Antonio J. Peña and Marc Jorda for as they proved an unlimited source of help. I am also grateful to the SoHPC coordinator, Mr. Leon Kos for his constant presence and guidance when I needed him. Finally my deepest gratitude to my colleague and friend Perry Gibson for his humorous comments and advice that made this experience unique.

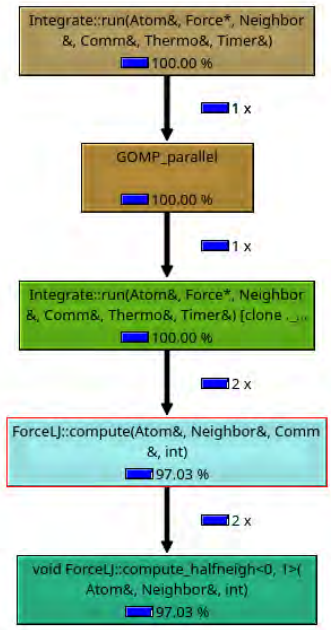
PRACE SoHPCProject ID

1901

Exploring Reproducible Workflows in Automated Heterogeneous Memory Data Distribution Literature Results

“It worked on my machine... 5 years ago”

Perry Gibson



Leveraging emerging heterogeneous memory subsystems for memory bound HPC applications requires an understanding of memory access patterns of the target workload. We revisit prior work from 2014, which simulated cache behaviour of HPC test workloads, and produced near-optimal data placement strategies from high-fidelity runtime data. We find that the results of the original simulations cannot be precisely reproduced. We document our approach of attempting to reproduce the results, and put forward concrete solutions to improve the reproducibility of future HPC research endeavours.

HPC applications the de facto expensive workload. At scale, even sub-percentage acceleration can represent massive cost savings. Thus, opportunities across the stack must be exploited as much as possible. Often the biggest bottleneck in these applications is data access times. If data is stored close to the processor when needed, such as at the lowest level of cache, then we incur very little overhead. However, this isn't feasible in all cases, given typical sizes of L1 caches are on the order of a dozen or so KiB, and higher levels of cache are on the order of a few dozen MiB. Since even modest HPC applications use GiBs of data, or orders of magnitude more, we are forced to store it in main memory, or disk.

A promising mitigation for this problem is moving to a heterogeneous memory model. Instead of the classic hierarchical memory model of *L1 cache* → *L2 cache* → ... → *LL cache* → *Main Memory* → *Disk*, we have a choice to place individual data in a number of memory subsystems with different properties. To prepare current and future HPC applications to exploit the potential benefits of this change, it is necessary to devise novel approaches for placing data objects in an appropriate memory subsystem, giving good automatic placement with minimal programmer involvement, while providing fine grain control if needed.

To perform this profiling, in 2015 researchers at BSC heavily adapted the Valgrind instrumentation framework,[1] so that they could measure the access patterns for individual data objects in a running program. The fork, developed for a suite of papers from the BSC, is described in more detail in.[2] The main contribution was adding the ability to trace loads and stores to individual variables, and in particular dynamic objects which are created by a number of lines of code, and are thus less easy to detect directly (for example arrays of pointers).

The results of using this system, named EVOP (Extended Valgrind for Object-differentiated Profiling), on two HPC mini-applications[4] [1] were presented in.[5] The paper demonstrated the scope for acceleration, which is still true today. However, the exact experimental results of the access patterns for the candidate applications and resulting data placement proposal differ.

[1] Interested readers can learn more about mini-applications in the SoHPC podcast episode featuring Dr. Mike Heroux.

Methods

We began our work by getting access to the original EVOP codebase, and the candidate workloads. Initially, we used the latest version of the code, committed two years after the original paper. We sought to first understand its functionality and configuration, then began to run full experiments. With the latest version of the code, we found that for our first workload, miniMD, the number of executed instructions was 17.9% less than the original results, and the last-level cache miss rate was 64.1% higher. Care was taken to ensure that the cache configuration was identical to what was described in the original paper.

Our next step was to eliminate the possibility that some version change in EVOP had changed its instrumentation behaviour. Using version-control commit timestamps, we examined the differences between candidate code versions that may have been used for original results. However, none of these versions could be successfully compiled. For commits of interest, source code files used inconsistent API calls, and we hypothesise that during development not all of the working files were tracked correctly. We patched a number of these code versions, and found identical results. This was promising for the consistency of the tool across versions, but did not explain the difference with the original paper.

We hypothesised that the compiler version used to produce the workload binaries, coupled with versions of the standard library could be an influence. Thus, we used a containerisation workflow (via the HPC-first Singularity system[6]) to systematically compare candidate environments. Many of these older tool-versions no longer existed in package repositories or archives, and required manual compilation. The effect of the compiler version on the number of instructions and data access patterns was found to be minimal. Variation of the instruction set extensions used by the compiler (such as (dis/en)abling vector instructions) changed the number of instructions, but not the data access patterns.

Consulting with the original authors, we reconstructed the original OS environment, using an archived distribution. This test for an unknown factor of influence. However, again the results remained identical.

With a reproduction of the original work appearing unlikely, we still consider the results we collected to be sensible. Thus, a useful extension to the work would be to compare the results of a simulate run against the results on actual hardware. We followed the approach of,[3] which used the Extrae framework developed at BSC. The use of Intel PEBS sampling would allow a similar type of object tracing information to be produced. The advantage over simulation is reduced overhead, at the cost of lower fidelity data. Collaborating with a colleague investigating the use of sampling in EVOP would enable a fair comparison. We had access to a configuration file believed to be used to gather the results in.[3] However, in the time remaining we were not able to collect the instrumented data of interest, namely object addresses and their access patterns.

Results and Conclusion

We have ruled out with some degree of certainty the influence of compiler and standard library versions. Because of the inconsistent version-control history, we cannot rule out that a different version of the codebase was used to perform instrumentation. However, our patched versions of EVOP generated the same results.

As one would expect, the instruction set architecture used has an influence on the number of instructions that a given program execution takes. However, it does not change the simulated cache behaviour. We followed the description of the cache configuration, and input parameters to our test workloads. However the exhibited behaviour was different. The simulator's behaviour should be agnostic to the underlying hardware it runs on, and we found this was true for small scale experiments.

In conclusion, the replication of results after years of tools lying dormant is a difficult one. The configurations and sequence of steps used to produce results is easily lost, and can be non-trivial to recover at a later date.

Our recommendation is that during the development stages of research tools, continuous integration systems are used to ensure that the versioned-project state is as expected.[7] We have created a continuous integration to be used in future versions of EVOP.

When the results for publications are being produced, then containerisation should be used to enshrine the configuration and tool version used to generate the needed computations. These details are generally not included in publications, as they are superfluous to the conceptual contribution of a work. However, it is clear that once lost to time they cannot easily be recovered, and for some systems are necessary for work's conclusions. This was not the case for EVOP, as our results using modern toolchains still trace objects and demonstrate theoretical speedups for candidate object distributions. They differ numerically from the original results. Ideally, containerised workflows should be published, but can also be kept in internal archives is aspects of their functioning are sensitive.

The pitfall of relying on external package repositories for the creation of containerised workflows should be heeded by researchers whose results are possibly influenced by such factors. Including the packages in the project archives should be considered a preferred practice.

References

[1] Nethercote, Nicholas and Seward, Julian. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation.

[2] Pena, Antonio J. and Balaji, Pavan. A Framework for Tracking Memory Accesses in Scientific Applications.

[3] Servat, H. and Peña, A. J. and Llort, G. and Mercadal, E. and Hoppe, H. and Labarta, J. (2017). Automating the Application Data Placement in Hybrid Memory Systems.

[4] Crozier, Paul Stewart and Thornquist, Heidi K. and Numrich, Robert W. and Williams, Alan B. and Edwards, Harold Carter and Keiter, Eric Richard and Rajan, Mahesh and Willenbring, James M. and Doerfler, Douglas W. and Heroux, Michael Allen Improving Performance via Mini-Applications.

[5] Peña, A. J. and Balaji, P. (2014). Toward the Efficient Use of Multiple Explicitly Managed Memory Subsystems.

[6] Kurtzer, Gregory M. and Sochat, Vanessa and Bauer, Michael W. Singularity: Scientific Containers for Mobility of Compute.

PRACE SoHPCProject Title

Reproducing Automated Heterogeneous Memory Data Distribution Literature Results and Beyond

PRACE SoHPCSite

BSC, Catalonia

PRACE SoHPCAuthors

Perry Gibson

PRACE SoHPCMentor

Peña, A. J.

Perry Gibson

PRACE SoHPCAcknowledgement

Many thanks to support and guidance from my mentor, the researchers and staff of the BSC, the organisers of the SoHPC at PRACE for getting things working and my cohort for bringing interesting stories to the weekly meetings. Special thanks to Dimitris Voulgaris for working closely with me throughout the project.

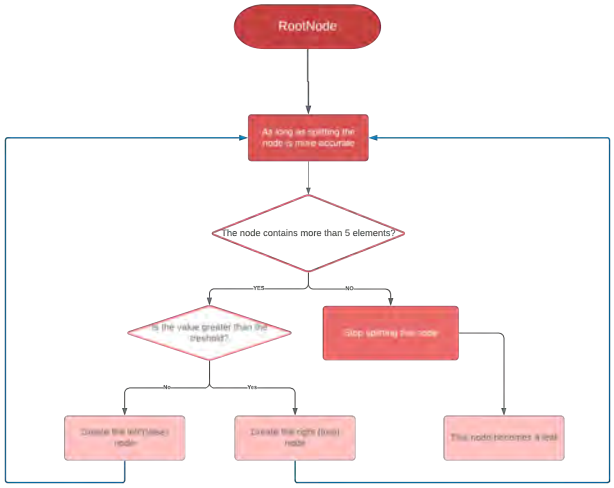
PRACE SoHPCProject ID

1902

What can we gain if we use HPC tools for Machine Learning instead of the JVM-based technologies?

High Performance Machine Learning

Thizirie Ould Amer



Machine Learning (ML) algorithms are designed to process and learn from **huge amounts of data** and their correlations to adapt and adjust the results. This big quantity of data needs to be analyzed and classified according to various criteria, which means that our **ML algorithms** have to evaluate these criteria and select the best ones. As you can imagine, this requires a lot of **time and energy** to be done. So how can we increase the efficiency of ML algorithm runs?

HPC is the solution for many kinds of problems! Combined with other features, HPC tools can have a big impact on the performance of our applications (programs). Big data processing is currently dominated by JVM-based technologies such as Hadoop MapReduce or Apache Spark. Their popular tools are indeed fairly efficient, and particularly, they are easy to use for developers. Our goal is to evaluate the traditional HPC tools such as MPI, OpenMP or GASPI for ML / big data processing, and to see for our self it they are at least as good as the aforementioned tools, or maybe even better!

In order to successfully carry out this project withing a limited time frame, we chose one popular ML algorithm – (Gradient Boosting Decision Trees).

This algorithm is very popular in Kaggle competitions. Indeed, many of the biggest winners of these contests include Gradient Boosting algorithms in their winning models ensembles, amongst a variety of other models. The most commonly used library is XGBoost and it aims to provide a *Scalable, Portable and Distributed Gradient Boosting (GBM, GBRT, GBDT) Library*.[1]

The main focus of this project is then the GBDT but the other methods are as popular and successful as this one.

Decision Trees

The idea of the Decision trees algorithm is to find a set of criteria that will help us "predict" (classify or inter/extrapolate variables we pick.

In the example in the Table 1, let's see if we can predict if an algorithm can be parallelized or not.

Table 1: Example 1

IndepParts	LoopsDep	Parallel?
1	0	1
0	1	0
0	0	0

Since the goal is to predict the last variable (*Parallel?*), our decision tree will have to consider each feature (column) and see which one helps us the most to distinguish if a code can be parallelized or not. Once such feature is identified, we split the codes according to that variable. Selecting any feature can naturally lead to misclassification of elements, otherwise the feature on its own would correlate perfectly with the result. This misclassification is also referred to as the **impurity** – an error rate parameter ranging from 0 to 1; 0 for pure (no misclassification) and 1 for a group where every element is misclassified. In each step of the construction of the decision tree, we pick the variable which leads to split with the lowest impurity. Decision trees on their own tend to suffer from overfitting, so we must use additional tricks to make it a good ML method.

Gradient Boosted Decision Trees

Gradient boosting in general is an approach of creating a predicting model in the form of an ensemble of weak prediction models. In Gradient Boosting Decision Tree, the weak prediction model is, unsurprisingly, the Decision Trees. By fitting a decision tree to pseudo-residuals (i.e. errors) of another tree, we can significantly increase the accuracy of the data representation.

Typically, a limited side decision trees (of depth 4 to 8) are iterated multiple (typically less than 100) times, creating (at most) 100 decision trees that have different decision criteria and "fixing" at each iteration the errors that the previous tree has made. The algorithm doesn't give more weight to any of the trees but the tree number *i* focuses on individuals that didn't fit into their leaves/groups in the tree number *i-1* (i.e. focuses on the individuals that have the most important pseudo-residual values).

How to parallelize this?

When it comes to the concept of a parallel version of a program, many aspects have to be taken into account, such as the dependency between various parts of the code, balancing the work load on the workers, etc.

This means that we cannot parallelize the gradient boosted algorithm's loop directly since each iteration requires the result obtained in the previous one. However, it is possible to create a parallel version of the decision trees building code.

The algorithm for building Decision Trees can be parallelized in many ways. Each of the approaches can potentially be the most efficient one depending on the size of data, number of features, number of parallel workers, data distribution, etc. Let us discuss three viable ways of parallelizing the algorithm, and explain their advantages and drawbacks.

Sharing the tree nodes creation among the parallel processes at each level

The way we implemented the decision tree allows us to split the effort among the parallel tasks. This means that we would end up with task distribution schematically depicted in the Figure 1.

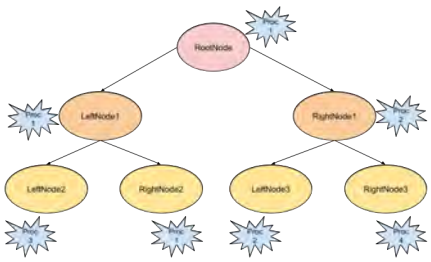


Figure 1: Parallelizing the nodes building at each level. Levels are in different colours.

Yet, we have a problem with this approach. We can imagine a case where we would have 50 "individuals" going to the left node, and 380 going to the right one. We will then expect that one processor will process the data of 50 individuals and the other one will process the data of 380. This is not a balanced distribution of the work, some processors doing nothing while others maybe drowning in work. Furthermore, the number of tree nodes that can be processed in parallel limits the maximum number of utilizable parallel processes... So we thought about another way.

Sharing the best split research in each node.

In our implementation of the decision tree algorithm, there is a function that finds the value that splits the data into groups with minimum *impurity*. It iterates (for a fixed column – variable) through all the different values and calculates the impurity for the split. The output is the value that reaches the minimum impurity.

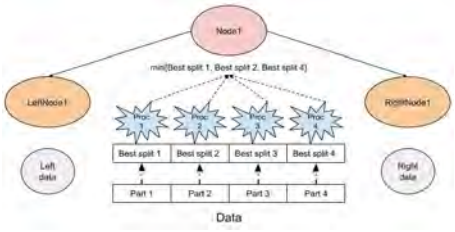


Figure 2: Sharing the best split research in each node, for each feature.

As can be seen in the Figure 2, this is the part of the code that can be parallelized. So every time a node has to find a split of individuals in (two) groups, many processors will compute the best local splitting value, and we keep the minimum value from the parallel tasks. Then, the same calculations are repeated for the *Right data* on one side, and for the *Left Data* on the other.

In this case, a parallel process will do its job for a node and when done it can directly move to another task on another node. So, we got rid of the unbalanced workload since (almost) all processes will constantly be given tasks to do.

Nevertheless, this also has an notable drawback. The cost in communication is not always worth the effort. Imagine the last tree level where we would have only a few individuals in each node. The cost of the communication in both ways (the data has to be given to each process, and received the output at the end). The communication will eventually slow down the global execution more than the parallelization of the workload speeds it up.

Parallelize the best split research on each level by features

The existing literature[2] helped us to merge some features of the two aforementioned approaches to find one that reduces or eliminates their drawbacks. This time, the idea is to parallelize the function that finds the best split, but for each tree level. Each parallel process calculates the impurity for a particular variable across all nodes within the level of the tree.
This method is expected to work better because:

1. The workload is now balanced because each parallel process evaluates the same amount of data for "its feature". As long as the number of features is the same (or greater, ideally much larger) than the number of parallel tasks, none of the processes will idle.

2. The impact of the problem we described in the second concept is reduced, since we are working on the whole levels rather than on a single node. Each parallel task is loaded with much larger (and equal) amounts of data, thus the communication overhead is less significant.

The global concept is shown in Figure 3

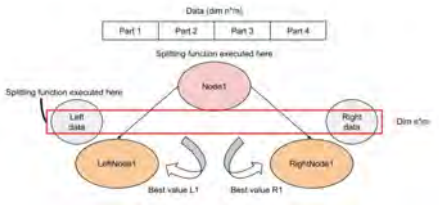


Figure 3: Parallelizing the best split research on each level by features

How can this be improved?

There are several ways to improve the performance of our parallel algorithm.

Let us focus on the communication, particularly its timing. While we cannot completely avoid loosing some wall-clock time in sending and receiving data, i.e. communication among the parallel processes, we can rearrange the algorithm (and use proper libraries) to facilitate overlap of the computation and communication. As shown in the the figure 4, in the first case (a) we notice that the processor waits until the communication is done to launch the calculations, whereas in the second case (b) it starts computation before the end of the communication process. The last case displayed in 5 shows different ways to facilitate total computation-communication overlap.

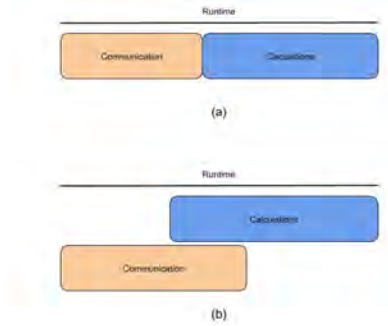


Figure 4: Timing of communication and computation steps: (a) no overlap, (b) partial overlap

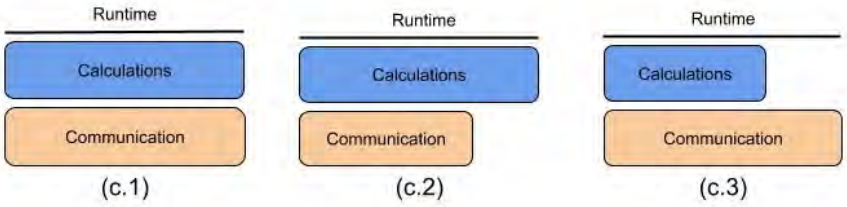


Figure 5: Timing of communication and computation steps: (c) complete overlaps.

One of the libraries that allows for asynchronous communications patterns, thus overlap of computation and communications is the GPI-2 (http://www.gpi-site.com), an open source reference implementation of the GASPI (Global Address Space Programming Interface, http://www.gaspi.de) standard, providing an API for C and C++. It provides non-blocking one-sided and collective operations with strong focuses on fault tolerance. Successful implementations is parallel matrix multiplication, K-means and TeraSort algorithms are discribed in Pitonak et al. (2019) "Optimization of Computationally and I/O Intense Patterns in Electronic Structure and Machine Learning Algorithms.".[3]

Conclusion

We have discussed some of the miscellaneous possible parallel versions of this algorithm and their efficiency. Unfortunately we did not have enough time to finalize the implementations and compare them with the JVM-based technologies but also with XGBoost performances.

Future works could focus on improving and continuing the implementations and comparing them to the usual tools. Including OpenMP in the parallelization could also be a very interesting approach and lead to a better performance.

Another side that could be considered as well is using the fault tolerant functions of GPI-2 which would ensure a better reliability for the application.

PRACE SoHPCProject Title
High Performance Machine Learning

PRACE SoHPCSite
Computing Center of SAS, Slovakia

PRACE SoHPCAuthors
Thizirie Ould Amer, Sorbonne University, France

PRACE SoHPCMentor
Michal Pitoňàk, CCSAS, Slovakia


Thizirie Ould Amer

PRACE SoHPCContact
Thizirie, Ould Amer, Sorbonne University
LinkedIn: in/Thizirie
E-mail: thizirie.ouldamer@gmail.com

PRACE SoHPCMore Information
Author's blog
About PRACE Summer of HPC

PRACE SoHPCAcknowledgement
First things first, I would like to thank my mentors Michal Pitoňàk and Mariàn Gall for their great help. Many thanks to my site coordinator; Lukáš Demovič and the whole CCSAS team who made my time here extraordinarily amazing. I would also like to thank my HPC and Data Analysis classes teachers; Pierre Fortin and Fanny Villers for their support.

PRACE SoHPCProject ID
1903

References

[1] XGBoost GitHub repository: https://github.com/dmlc/xgboost

[2] Parallel Gradient Boosting Decision Trees: http://zhanpengfang.github.io/418home.html

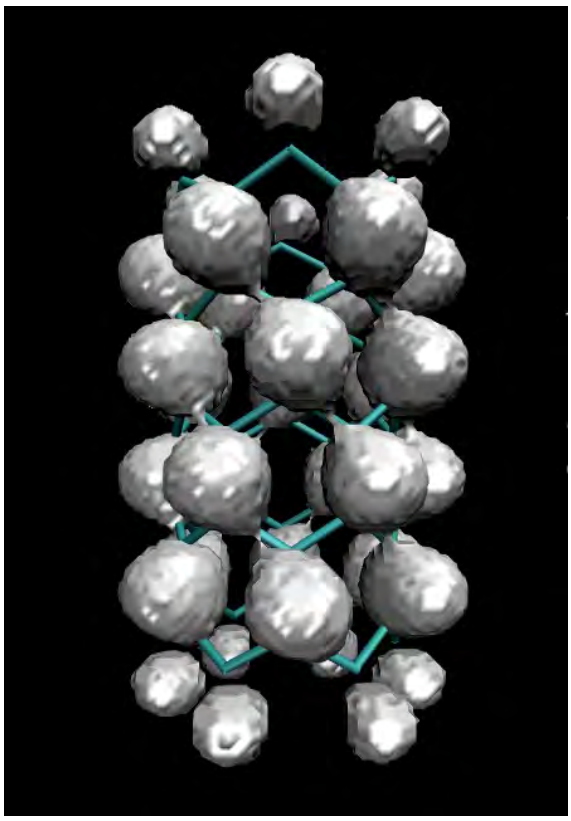
[3] Optimization of Computationally and I/O Intense Patterns in Electronic Structure and Machine Learning Algorithms: http://doi.org/10.5281/zenodo.2807938

[4] Gradient Boosting decision trees: https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

Where can you find the electrons of a nanotube?
Electron density computation for each orbital.

Electron density of Nanotubes

Irén Simkó



We exploit the helical symmetry of nanotubes in the electronic structure computation. The code was extended with electron density computation for each orbital. I used Message Passing Interface for parallelization and visualized the electron density isosurfaces.

Have you heard about (carbon) nanotubes? Probably you already had, but you will hear a lot more about them in the future. The extraordinary mechanical and electrical properties of nanotubes have a lot of promising applications in engineering and material science. For example, composite materials with carbon nanotubes have great mechanical strength. Nanotubes are candidates for nano-electronics as well, because they can behave like metals, semiconductors or insulators.

The topic of my project was simulating the nanotubes with the SOLID98 quantum chemical program.[1] The program –written in FORTRAN77– computes the electronic structure of the nanotubes.

According to the principles of quantum mechanics, the states of the electrons have well-defined discrete energies and wave functions (orbitals), which are computed by the program. The parallelization of the code was the task in the previous years of Summer of HPC, and my goal was to extend the code with a new feature: Electron density computation and visualization. This shows us the distribution of electrons in the space, to see the chemical bonds and the nodes of the orbitals.

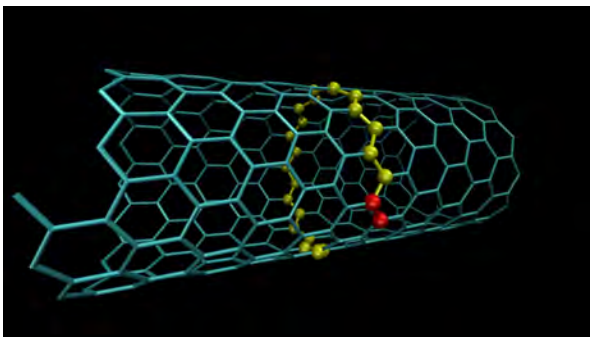


Figure 1: Helical symmetry of the nanotube

Use symmetry when you can!

The computational cost of a quantum chemical simulation strongly increases with the size of the system. But how can we still compute large systems such as nanotubes or crystals? One solution is periodicity and symmetry, which means that we can build the whole large system by replicating a small building block (unit cell) following symmetry rules. Since the replicas are not independent, the size of the unit cell determines the cost of the computation.

Nanotubes have helical symmetry[2] which allows us to use very small unit cells. For carbon nanotubes it is enough to have only two atoms! Then, imagine a spiral that is winding on the surface of a cylinder, and replicate the unit cell following the path of the spiral. This way you can build the whole carbon nanotube. Using helical symmetry is a much better approach than considering only the translational symmetry along the axis of the nanotube.

The electronic structure computation

To get the electronic orbitals and energies we have to solve the Schrödinger-equation. But do not grab a pen and paper, because it has an analytical solution only for simple systems, we have to do it numerically in other cases. The wave function is a linear combination

of basis functions in the simulation. The Schrödinger-equation is translated to an eigenvalue-equation of the so-called Fock matrix, which is solved iteratively.

The diagonalization part of the program is parallellized with Message Passing Interface (MPI). Thanks to some mathematical tricks the Fock matrix is block diagonal, eack block labelled with a k value and diagonalized by a different process. As a result, we get orbitals and energies for each k block.

Electron density: The serial code

The electron density is the probability of finding the electron at a given point is space. We would like to visualize the individual electron orbitals, so we restrict the computation for one orbital at a time.

Electron density computation is the last step of the program, we use the results of the last diagonalization. From the eigenvectors, we build the so-called density matrix (P), and contract it with the basis functions (χ). „Contraction" is a double sum over all basis functions of the system:

$$\rho(\mathbf{r}) = \sum_{a,b} P_{ab}\chi_a(\mathbf{r})\chi_b(\mathbf{r})$$

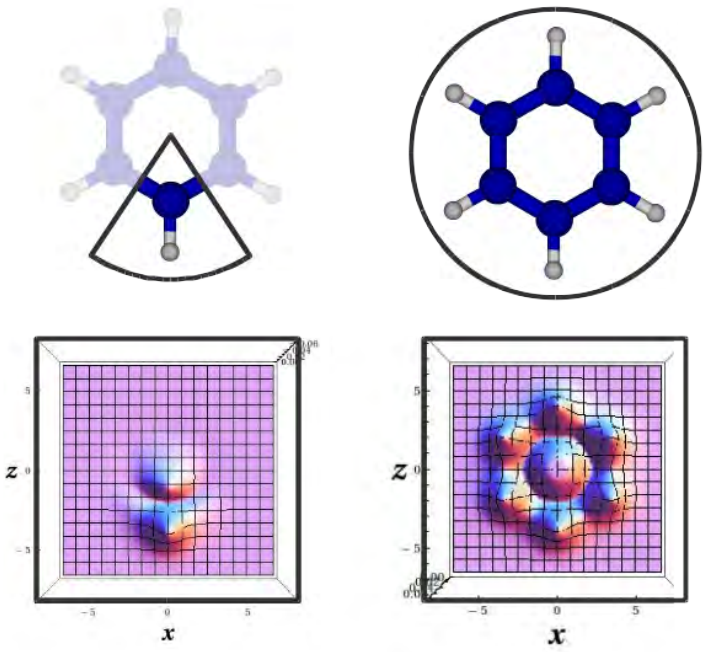


Figure 2: Electron density of one unit cell and the whole benzene molecule

First, we compute the contribution of the central unit cell of the electron density. Here in the sum we take one basis function from the central unit cell and the other one from the whole nanotube. Then we get the total electron density in the same manner as we built the nanotube from the unit cell by replicating the contribution of the central unit cell. In the figure below you can see an example for the benzene molecule.

Parallelization using MPI

Next step was to parallelize the electron density computation. I decided to use the same technique as it is in the diagonalization of the Fock matrix. We use the Master-Slave MPI system, where the Master process (rank = 0) distributes the work among the Slave processes (rank = 1,2,...,$N-1$). First the Master process does some initialisation and broadcasts the data to the Slaves, then each process does the work assigned to them.

The electron density computation is done in two nested loops: one loop over the k values and one loop over the orbitals of a given k. In the first version of the code, each process computes the orbitals belonging to a different k value. We use a counter variable to distribute the work. The counter is set to 0 at the beginning of the loop.

A process does the upcoming k value if its rank is equal to the actual counter value. If a process accepts a k value, it increases the counter value with 1, or updates it to 0 if it was $N-1$. This way, process 0 does $k=1$, proc. 1 does $k=2$, proc. $N-1$ does $k=N$, then proc 0 does $k=N+1$, and so on.

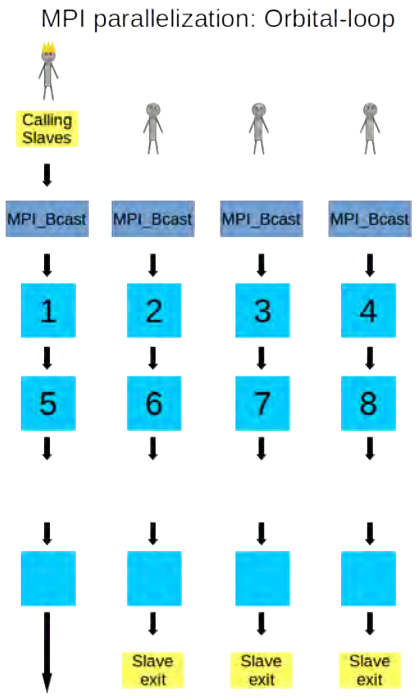


Figure 3: MPI parallelization

The problem with this implementation is that you cannot use more processors than than number of k values. So in the second version I moved the parallelization to the inner loop over the orbitals. If we have M orbitals in each block, the first M processes start working on the $k=1$ orbitals, the next process gets the first orbital of $k=2$. This is a much better solution, because we can utilize more processors than in the first case.

On the other hand, the diagonalization of the Fock matrix is still parallelized at the k-loop, so we have idle processors if we have more than the number of k values. The relative time of the diagonalization and the electron density computation determines if it is worth to use the k-loop or the orbital-loop parallelization.

"A picture is worth a thousand words"

One goal of the project was to actually *see* where we can find the electrons, but visualization turned out to be more challenging than I expected. A good scientific figure should capture the essence of the topic of the research as well as catch the attention of the reader.

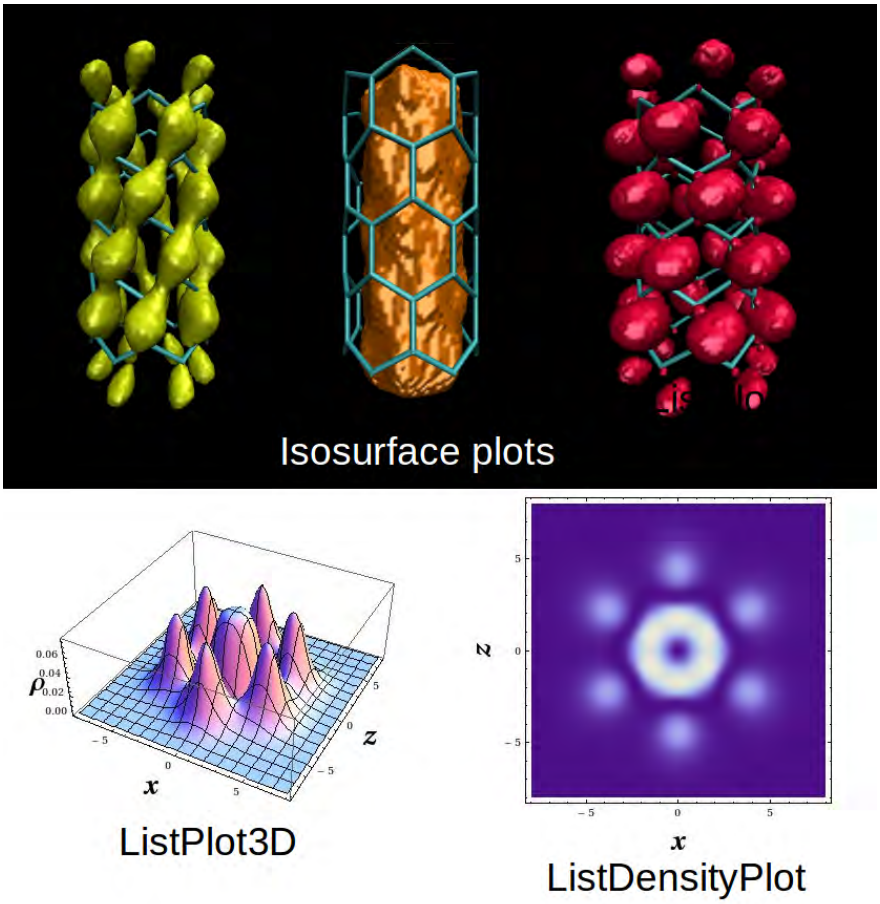


Figure 4: Electron density visualization examples

The result of the computation is the electron density in a large number of grid points. If the grid points are on a plane, I plotted the results with Wolfram Mathematica using ListPlot3D and ListDensityPlot. On the figures above you can see an example for a benzene orbital, with the electron density calculated at the plane of the molecule.

I visualized the electron density in space by plotting the isosurface, which is a surface where the electron density equals to a given value. I used the Visual Molecular Dynamics program to make isosurface plots after transforming the output to Gaussian cube file format.

Results & discussion

I tested the electron density computation and visualization for the benzene molecule and a small carbon nanotube. I tried to find the physical meaning of each electron density plot. For example, there are bonding benzene orbitals where the electron density is high between the atoms; and there are antibonding orbitals with nodal surface, where the electron density is zero. In the case of some nanotube orbitals the electron density is localised inside the tube, while in other you can see the bonds between the carbon atoms.

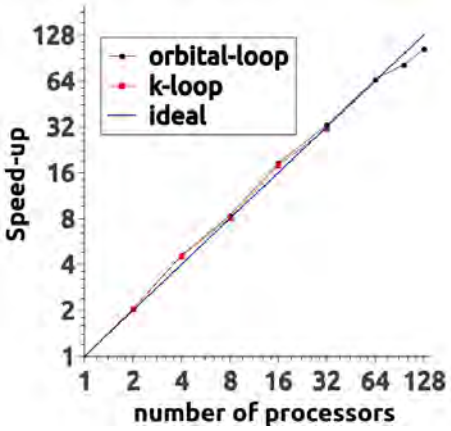


Figure 5: Speed-up of the parallel program

As for the parallelization, I made a test for a nanotube that has 32 k values. In the figure above, you can see the speed-up for the two code versions. The maximal number of cores was 32 for the k-loop version, and 128 for the orbital-loop version. Up to about 64 cores the speed-up is ideal, but after that the program will not be much faster if we increase the number of cores.

References

[1] *J. Comp. Chem.*, Vol. 20, No. 2, 253.261 (1999)

[2] *J. Phys. Chem. B* 2005, 109, 52-65

PRACE SoHPCProject Title

Electronic structure of nanotubes by utilizing the helical symmetry properties: The code optimization

PRACE SoHPCSite

Computing Centre of the Slovak Academy of Sciences, Slovakia

PRACE SoHPCAuthors

Irén Simkó, Eötvös Loránd University, Hungary


Irén Simkó

PRACE SoHPCMentor

Prof. Dr. Jozef Noga, DrSc., CCSAS, Slovakia

PRACE SoHPCContact

Irén, Simkó
E-mail: simkoiren@freemail.hu

PRACE SoHPCSoftware applied

SOLID98, Visual Molecular Dynamics, Wolfram Mathematica, Blender

PRACE SoHPCAcknowledgement

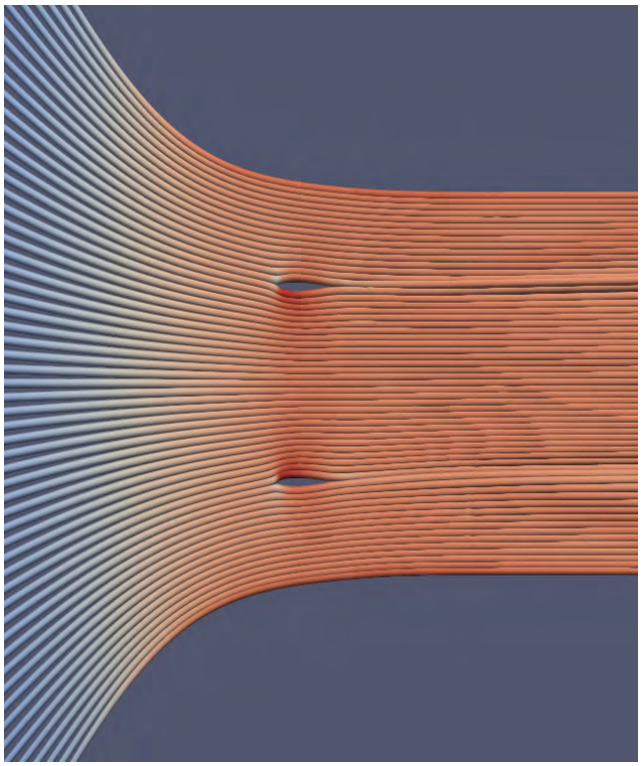
I am grateful to my mentor, Jozef Noga, my site co-ordinator, Lukáš Demovič, and all people at CCSAS for their support and the wonderful summer. Calculations were performed in the Computing Centre of the Slovak Academy of Sciences using the supercomputing infrastructure acquired in project ITMS 26230120002 and 26210120002 (Slovak infrastructure for high-performance computing) supported by the Research & Development Operational Programme funded by the ERDF.

PRACE SoHPCProject ID

1904

In Situ / Web Visualisation of CFD Data using OpenFOAM

Li Juan Chan



Paraview Catalyst has been proposed to overcome the limitations caused by I/O bottleneck. In this research, the scalability of OpenFOAM with coprocessing has been investigated and it has shown to be very scalable.

Over the years, the computational power of processor is developed in a tremendous fast pace. This has made large-scale computing to become more affordable. The development of parallel computing software is advancing equally as fast as the processor. However, there are subsystems of high-performance computer that are not developed at the same pace with processor. One of them is the I/O bandwidth. I/O is developed in such a slow pace that it is now becoming the bottleneck for exascale computing. This project hoped to address this issue and try to get around the bottleneck by using in situ visualisation.

Traditionally, the simulation process consists of pre-processing, processing and post-processing. However, this process is getting more and more expensive due to the I/O bottleneck. Writing and reading data have become very slow when compared to the processing speed. In situ visualisation was designed to solve this problem by moving some of the post-processing tasks in line with the simulation code.

Paraview has come out with a in situ visualisation library called Catalyst. Catalyst enables live visualisation of simulation while the simulation is still running. There are a few benefits of using Paraview Catalyst. One of them is that the live visualisation feature of Paraview Catalyst enables researchers to examine whether the pre-processing step is set up correctly. If the boundary conditions are found to be incorrect, the simulation can be stopped immediately before investing more time and resources into an incorrect simulation. Furthermore, with Paraview Catalyst, the frequency of data writing is significantly reduced as only the final result is needed to be saved. The intermediate result can be viewed instantly without having to be saved for post-processing.

Therefore, one may wonder if Paraview Catalyst will increase the demand of memory due to live visualisation. The answer is yes and no. The live visualisation definitely requires some memory to run. However, the demand of memory is not significant because Paraview has the ability to extract a small amount of important data from the result instead of saving the full datasets. The features that can be extracted from Paraview include slice, clip, glyphs, streamline, etc.

Aim and Motivation

In the field of high-performance computing, the scalability of a software is very important. This is because supercomputer is nothing but a bunch of computers combined together to perform a task. Therefore, researchers may want to know how much benefit can be obtained by running a software with additional resources. Since Paraview Catalyst is so useful and may be widely used in the future, I, as a researcher, would like to find out how scalable Paraview Catalyst is and this basically is the aim of this project.

How to use Paraview Catalyst?

After introducing Paraview Catalyst, I would like to introduce another software that is used together with Paraview Catalyst, which is OpenFOAM. OpenFOAM is an open-source software for computational fluid dynamics (CFD) and it is also highly scalable. Therefore, the scalability of OpenFOAM with and without co-processing will be determined and compared.

The model I was working on is shown in Figure 1. It is a wind tunnel construction with two NACA0012 airfoils, rotating with respect to hinges.

Figure 2 is the zoomed in image of the model. As shown in the figure, structured mesh is used in the upstream and downstream of the model.

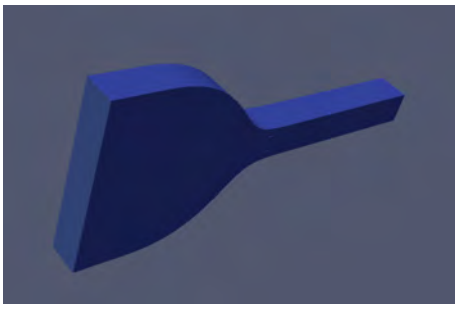


Figure 1: Model

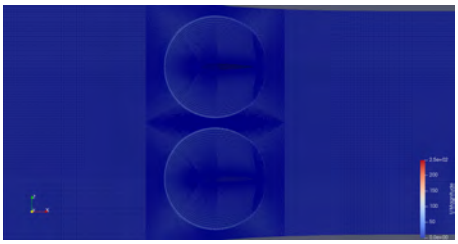


Figure 2: Mesh

The mesh in the middle is unstructured due to the rotation of the airfoils. There are approximately 10 million cells in the mesh. A compressible solver named rhoPimpleFoam is used for all simulations.

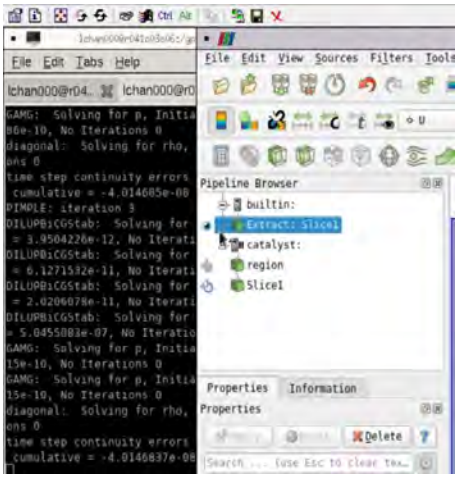


Figure 3: Live Visualisation

To run OpenFOAM and Paraview Catalyst at the same time, a software called Remote Connection Manager (RCM) is used. It allows the users to connect to the compute node of the supercomputer. One RCM session is used to run the Paraview. When Paraview is loaded, Catalyst can be connected from the menu bar of the Paraview. Meanwhile, another RCM session is opened to run OpenFOAM. However, an OpenFOAM plugin, co-developed by CINECA with ESI-OpenCFD and Kitware, should be loaded before running OpenFOAM. Once loaded, the simulation can be run and the intermediate result of the simulation can be visualised concurrently as shown in Figure 3.

In Figure 3, the feature displayed is a slice. To extract other types of features, a different pipeline is needed. A pipeline is a python script that specifies the feature to be extracted and the parameters of the feature. The pipeline can be obtained from Paraview GUI. The process is repeated with different number of cores and different pipelines. In this study, several pipelines, for example slice, clip, glyphs, streamline and region are produced as shown in Figures 4, 5, 6,7 and 8.

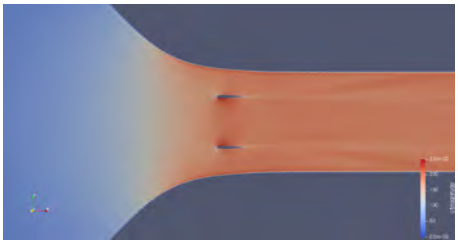


Figure 4: Slice

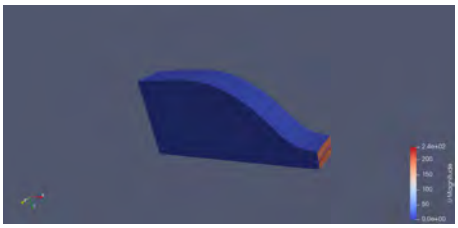


Figure 5: Clip

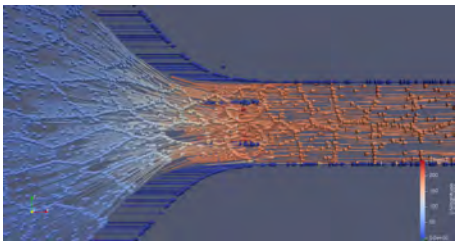


Figure 6: Glyphs

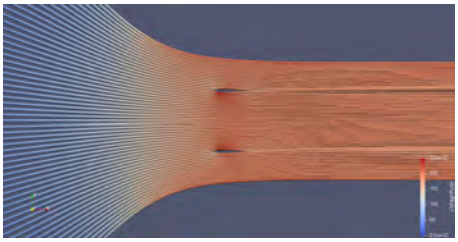


Figure 7: Streamline

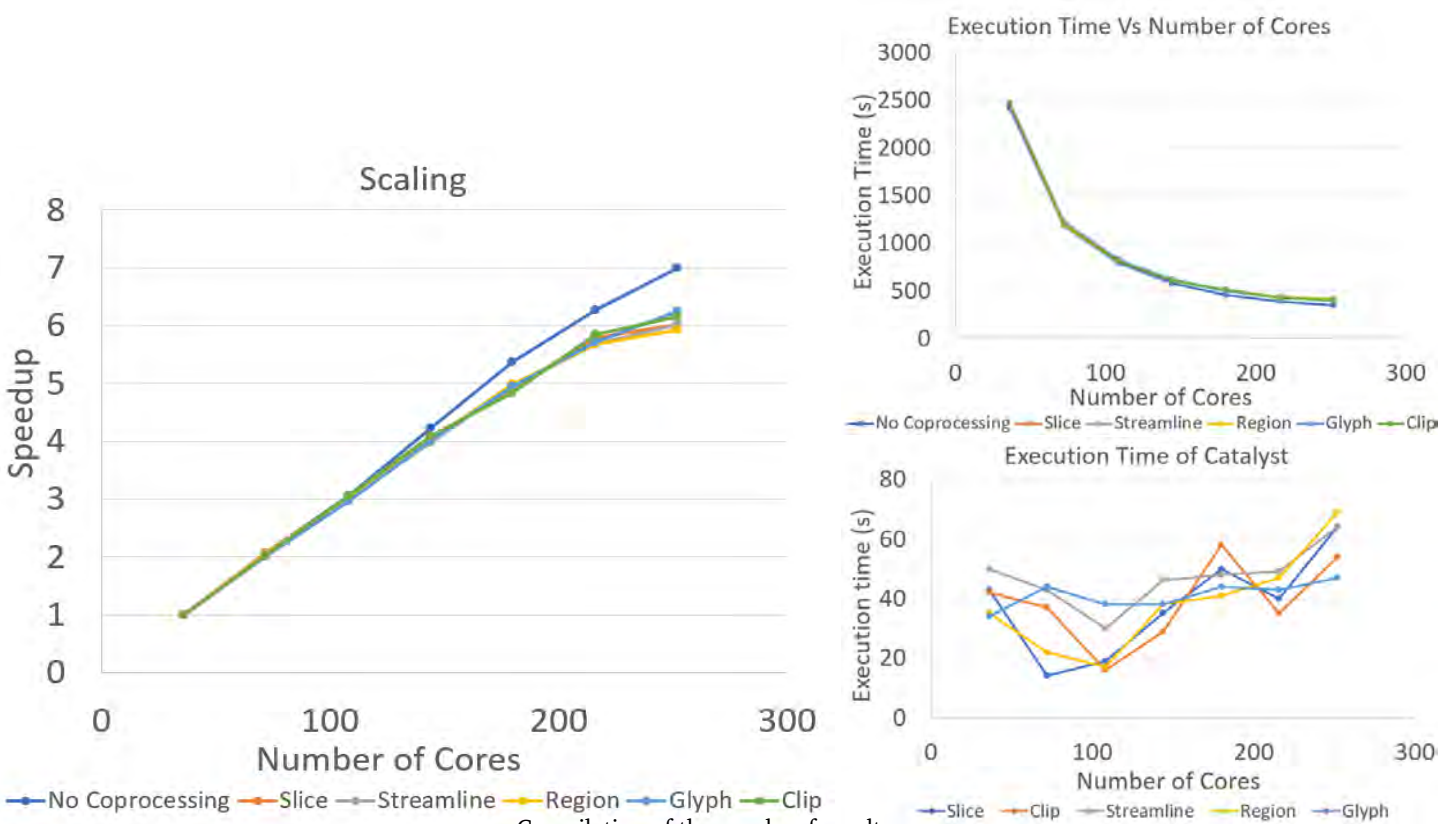
Figure 8: Region

What is the outcome?

The three graphs in the next page are to summarise the results of this research. It consists of scaling graph, graph of execution time of OpenFOAM with and without catalyst and graph of execution time of Paraview Catalyst.

First of all, what is speedup? Speedup is a variable that measures scalability. The speedup of n cores is defined as the execution time in one core divided by the execution time in n cores, where n is the number of cores to be measured. Basically, it measures the relative performance of one core and n cores processing the same problem. In the scaling graph, one may notice that the speedup of OpenFOAM with coprocessing are very similar regardless of the type of pipeline. Additionally, the speedup of OpenFOAM with and without coprocessing are very similar up until 108 number of cores. Beyond that point, they start to diverge with the OpenFOAM without coprocessing being more scalable than that with coprocessing. The degree of divergence increases as the number of cores increase. We also noticed that the scalability of OpenFOAM with and without coprocessing starts to flatten out beyond 216 number of cores. The sudden reduction of their scalability is due to communication overhead. Therefore, if a good scaling after 216 cores is wanted, the size of the mesh is needed to be increased in order to have a good balance between computation and communication,

Similar to the scaling graph, the execution time of OpenFOAM with and without coprocessing are very similar. However, the execution time without coprocessing is slightly faster at large number of processor cores. The difference increases as the number of processor cores increase. In contrast, the execution time of Paraview Catalyst do not seem to have a clear pattern as the two previous graphs. However, all pipelines exhibit a trend, in which the execution time decreases up until 108 processor



Compilation of the graphs of result

cores. Beyond that point, the execution time increases. This trend agrees with the scaling graph.

In a nutshell

OpenFOAM with Paraview Catalyst is very scalable with only slightly inferior to pure OpenFOAM. However, the benefits that can be obtained from Paraview Catalyst definitely outweigh the slight reduction of scalability.

For anyone who is interested in this project, the details of this project can be found in the github repository.[1]

References

[1] https://github.com/ljchan1/SoHPC_19

PRACE SoHPCProject Title

In Situ / Web Visualisation of CFD Data Using OpenFOAM

PRACE SoHPCSite

CINECA, Italy

PRACE SoHPCAuthors

Li Juan Chan, University of Manchester, UK

Li Juan Chan

PRACE SoHPCMentor

Prof. Federico Piscaglia, Dept. of Aerospace Sci. and Technology Politecnico di Milano, Italy
Simone Bnà, CINECA, Italy

PRACE SoHPCContact

Li Juan Chan, University of Manchester
Phone: +60 125271362
E-mail: lijuan.chan@hotmail.com

PRACE SoHPCSoftware applied

OpenFOAM 1806, Paraview 5.5.1, Catalyst, Blender 2.79b

PRACE SoHPCMore Information

www.paraview.org
www.openfoam.org

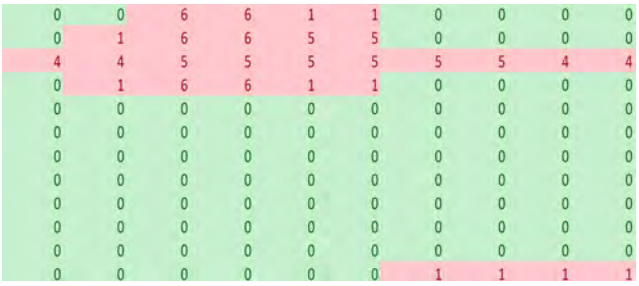
PRACE SoHPCAcknowledgement

I would like to express my utmost gratitude to my site coordinator, Simone Bnà for providing me the guidance and advice throughout the project. Without his supervision, this project would never reach as far as it has. Furthermore, I would also like to thank PRACE for sponsoring me to work in CINECA. I also like to thank Massimiliano Guarrasi and Leon Kos for arranging my allocation in Bologna. Last but not least, I would like to thank Prof. Federico Piscaglia and Federico Ghioldi for providing me the model.

PRACE SoHPCProject ID

1905

Explaining the faults of HPC systems



Martin Molan

This work provides initial investigation into possibilities for machine learning on log data collected on HPC systems at CINECA supercomputer site. General data transformation/preparation pipeline that can be the basis for further machine learning projects is presented. Additionally, this work also explores possibilities for creation of explainable machine learning models on processed data.

The goal of presented work is to explore possibilities for machine learning on top of data collected from high performance computing (HPC) systems. The data was collected on two HPC systems (MARCONI and GALILEO) from supercomputer site CINECA.

Presented work has the following key points:

- Preparation of dataset for future machine learning experiments. This involves creation of data transformation/pre-processing pipeline that is capable of handling huge quantities of data in parallel and in batches.
- Building of explainable machine learning (ML) model that investigates a learning task on a subset of data
- Implementing scalable parallel induction of *stable* decision trees.

1 Dataset preparation

The main goal of dataset preparation is to convert data from raw data (each row represents single attribute, single time stamp and single node) to data appropriate for machine learning. The processing should be done in parallel and in batches.

1.1 Feature construction and dataset description

The dataset has the following attributes (features):

```
alive::ping
backup::local::status
cluster::status::availability
cluster::status::criticality
cluster::status::internal
dev::raid::status
dev::swc::confcheckself
filesys::local::avail
filesys::local::mount
filesys::shared::mount
memory::phys::total
ssh::daemon
sys::ldapsrv::status
batchs::JobsH
filesys::eurofusion::mount
sys::gpfs::status
dev::ipmi::events
dev::swc::bntfru
dev::swc::bnthealth
dev::swc::bnttemp
dev::swc::confcheck
batchs::client::state
batchs::client
net::opa::pciwidth
net::opa
sys::orphaned-cgroups::count
core::total
sys::cpus::freq
batchs::client::serverrespond
MaxState
```

The only constructed feature is MaxState which is the value of most serious warning in a timestamp. All features have values between 0 and 3 where 0 means normal operation, 1 means warning and 2 means serious failure (3 signifies missing data).

Feature evaluation Informativeness of features is estimated by evaluating them with random foresters and extra tree classifiers (ensemble classifiers based on decision trees). The most relevant features evaluated are (according to both methods of evaluation):

```
batchs::client::state
MaxState
ssh::daemon
alive::ping
```

2 Supervised learning

The second goal of the project is to construct a supervised learning task. The training set consists of instances, described by attributes and a target value for each instance. In this work target value will always be discrete - the training task will be classification.

2.1 Comparison of different classification algorithms

Comparison of classification algorithms is done with 10-fold cross validation. In each repetition of the process, the model is trained on 90% of the data and evaluated on 10%. The results are then averaged across (10) iterations. 10-fold cross validation generally produces more reliable results than simple train/test split (it avoids overly pessimistic or optimistic estimates of classifier performance). The goal of the supervised learning model is to predict when a fault (a warning, or a fault) from a component (attributes described above) is serious enough to warrant a shutdown of a node (binary target class). Such (explainable) model would enable the operators to make more informed decision about the seriousness of the warnings provided by the system.

Table 1: Average classification accuracy in 10-fold cross validation

Dummy Classifier	0.9663
Nearest Neighbors	0.9865
Linear SVM	0.9926
RBF SVM	0.9924
Decision Tree	0.9928
Random Forest	0.9744
Neural Net	0.9915
AdaBoost	0.9928
Naive Bayes	0.9936

Table 2: Average gain compared to dummy classifier in 10-fold cross validation (in percentages)

DDummy Classifier	0.0
Nearest Neighbors	2.015
Linear SVM	2.625
RBF SVM	2.612
Decision Tree	2.644
Random Forest	0.805
Neural Net	2.514
AdaBoost	2.644
Naive Bayes	2.73

Classifier evaluation (and base classifiers themselves) are based on Scikit-leran library.[1] Each of iterations was performed in its own thread (parallelization).

The most significant takeaway form classifier performance is the relative strength of decision trees. This suggests that:

- Features are informative. Feature set provides good information about the target label without the need for additional feature construction.
- Features do not require non-linear transformations to be informative (SVMs do not represent improvement over base-line features

Informative features are the basis for creation of an explainable model. If that was not the case - if for instance SVMs performed considerably better than baseline trees - features would need non-linear transformations which would make them much more difficult to interpret.

Decision trees - in their basic implementation (greedy splitting) - are not a reliable explainable model for the problem of this scale. Because they are generally unstable (can significantly change with small changes on the dataset) reasoning provided by the decision trees cannot be the basis for the final model.

3 Future work

Based on the relatively good performance of decision trees the next step in building of the predictive model is to solve inherent instability of decision trees. A possible approach is a consensus tree classifier. This is an ensemble learning approach based on decision trees. It essentially combines predictions of several decision trees into a single (stable) model.

Basic algorithm for consensus tree induction can be summarized as:[2]

1. Perform N-fold partition of the data set (same partition as with cross validation), train separate decision tree on each partition
2. Compute dissimilarity matrix for instances with regards to the trained decision trees
3. Construct hierarchical clustering and clean the dataset
4. Construct decision tree on a cleaned data set

3.1 Implemented on summer of HPC

Preliminary experiments with parallel implementation (computation of dissimilarity matrix is parallel) of consensus tree algorithm were performed. The open problem remains how to efficiently implement the parallel version of the algorithm that would be applicable to bigger datasets (great number of instances and base estimators). The main problem of this implementation is the memory demands of each individual thread. Parallel implementation and subsequent consensus trees will be a topic of future paper prepared with mentors from university of Bologna.

References

[1] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E (2011). Scikit-learn: Machine Learning in Python *Journal of Machine Learning Research*, 12: 2825–2830

[2] Kavsek, B. and Lavrac, N. and Freligoj, A. (2001) Consensus Decision Trees: Using Consensus Hierarchical Clustering for Data Relabelling and Reduction *European Conference on Machine Learning*, 251-262

PRACE SoHPCProject Title

Anomaly detection of system failures on HPC machines using Machine Learning Techniques

PRACE SoHPCSite

CINECA, Bologna

PRACE SoHPCAuthors

Martin Molan, [International Postgraduate school Jozef Stefan Institute] Slovenia

PRACE SoHPCMentor

Andrea Bartolini, University of Bologna, Italy

PRACE SoHPCMentor

Andrea Borghesi, University of Bologna, Italy

PRACE SoHPCMentor

Francesco Beneventi, University of Bologna, Italy

Martin Molan

PRACE SoHPCContact

Martin, Molan, International postgraduate school Jozef Stefan Institute
E-mail: martin.molan@comtrade.com

PRACE SoHPCAcknowledgement

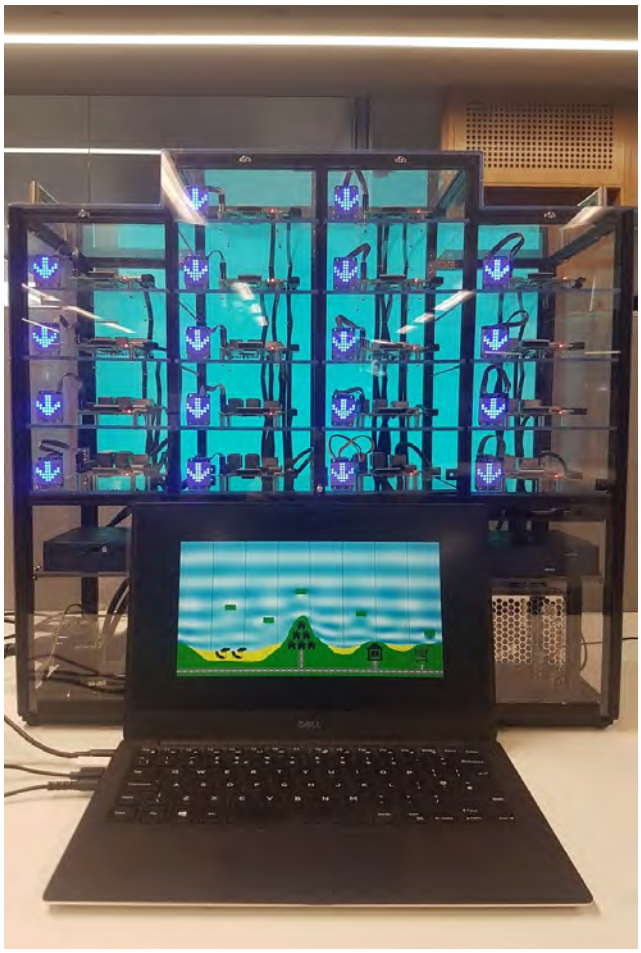
I am thankful to my mentor (dr. Bartolini) and his colleagues from university of Bologna for all the help and guidance. I am also thankful to my site supervisor dr. Massimiliano Guarassi for his help with setting up the environment on HPC.

PRACE SoHPCProject ID

1906

Parallel Computing Demos on Wee Archie

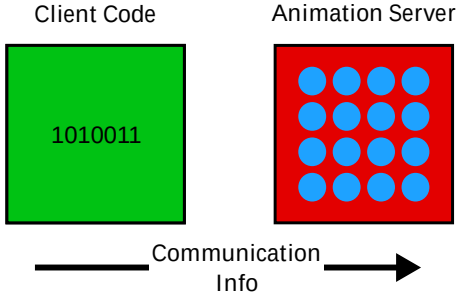
Caelen Feller



I created a framework for visualising parallel communications on a Raspberry-Pi based parallel computer Wee Archie. I used this to create demonstrations illustrating basic parallel concepts, and an interactive coastal defence simulation.

EPCC has developed a small, portable Raspberry-Pi based "supercomputer" which is taken to schools, science festivals etc. to illustrate how parallel computers work. It is called Wee Archie because it is a smaller version of the UK national supercomputer ARCHER. It is actually a standard Linux cluster and it is straightforward to port C, C++ and Fortran code to it. There are already a number of demonstrators which run on Wee Archie that demonstrate the usefulness of running demonstrations on a parallel computer, but they do not specifically demonstrate how parallel computing works.

In this project, I developed a framework for creating and enhancing new, existing and in-development demonstrations that show more explicitly how a parallel program runs. This is done by showing a real-time visualisation on the front-end of Wee Archie, or by programming the LED lights attached to each of the 16 Wee Archie Pis to indicate when communication is taking place and where it is going (e.g. by displaying arrows). All of these visualisations are triggered via the MPI (Message Passing Interface) profiling interface, a standard feature on all modern HPC systems, making this a drop-in solution for most existing code.



I developed visualisations for a tutorial illustrating the basics of parallel communications and a coastline defence demonstration. I also developed a web interface for the tutorials to create a more cohesive user experience. In these demonstrations, I aimed to be able to make it clear what is happening on the computer to a general audience.

Animation Server

To display communications via the LED panels (created by Adafruit), I used the official Adafruit Python library. As such, my visualisations are done in Python.

To allow all demonstrations to share the panels safely, I start a queue on each Pi on a separate background process, where any demo can add an 8px $\times$ 8px image to be displayed on the 8×8 LED panels. I can also give these images properties such as how long to be

displayed and create parcels of images together which form my various animations.

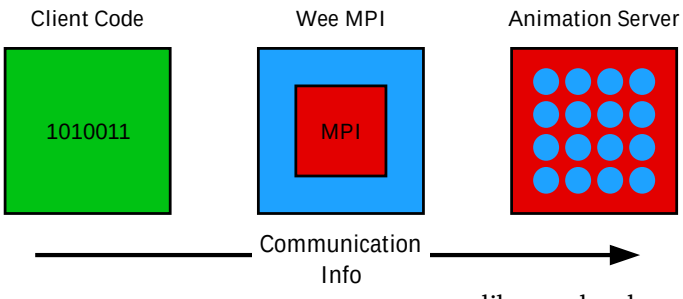
I had two major requirements from my animation system. I need to allow demonstrations developed in any programming language to create an animation, and to be able to coordinate animations between Pis. To do this, I made a web server on each Pi using Flask which will place an animation in queue when you make a certain web request against it. You must provide options for animation length, type, and type-specific options as discussed below.

Point to Point Visualisations

In MPI, an important class of communications are “point to point” communications. These are when a message is passed from one node (here meaning a Raspberry Pi) to another directly. In its most basic form, a node will start to send to some destination, and wait until that destination has begun to receive from the correct source before beginning to transfer the message.

Given the ability to add a sequence of images to a queue, how can I accurately visualise a "send" and "receive" operation between two Pis? My approach was to use a Python “pipe”. When an animation is reached in queue, it will show an “entry” and stop, using the pipe to wait until the server allows it to continue. This allowed me to synchronise and wait on animations between Pis. This accurately replicates the behaviour of MPI messaging.

This behaviour is known as a “synchronous” or “blocking” send. There also exists a “non-blocking” send. The main difference between blocking and non-blocking communications is that using non-blocking communication, the Pi



will continue working, not waiting for the communication to start, and do the communication in the background. This is a more efficient but less safe form of communication, and I provided visualisation for it as it is commonly used.

Other differences are discussed in the MPI standard.[1]

Collective Visualisations

The next major class of MPI communications are “collectives” - when many nodes want to communicate others at once. Say I want to distribute some result to every node - this is done using a broadcast. According to the standard,[1] this will cause every node participating to wait until it has the correct output before continuing, though the exact timing varies.

For clarity and consistency, in collective communication *visualisations*, all participants wait until the communication is done overall before continuing. The implementation is similar to that of waiting in point to point.

There were several other types of communication I visualised (gathering, scattering and reduction), whose implementation is similar. With gathering, the result is collected from each node and stored on one. With scattering, the result on one node is split into small, even parts and distributed to many. With reduction, the result is gathered but an operation is done to it as it is collected such as a sum or product. For more details on these, see the MPI standard.[1]

C and Fortran Framework

While demonstrations for Wee Archie typically use Python for their user interface, they use the C and Fortran languages in computations for performance reasons. Thus, it was desirable to create a wrapper around MPI in these languages which will automatically let the animation server know when a a communication is started.

I did this using the MPI profiling interface. MPI internally refers to functions using “weak symbols”, which allows you to override the functions provided by the library and allows a library developer to call their visualisation and logging code. The framework includes visualisations for most MPI communications, all shown on the next page.

Application-Specific Animations

Often a demonstration will require unique animations, such as a context-appropriate “working” animation, or a visualisation of some communication at a higher level of abstraction than MPI, such as the “haloswaps” of the coastal defence demonstration. Here, the animation server can be contacted using a non-MPI process which directly contacts the server and requires modification of the source code.

Unified Web Interface

As these demonstrations are used for outreach, the surrounding narrative is important for audience engagement. To improve this aspect of the Wee Archie interface is an important aspect of explaining more complex concepts such as parallel communication. In previous demonstrations for Wee Archie, a framework written in Python is used to display a user interface on a connected computer. This starts the demonstration by contacting a demonstration web server running on Wee Archie, which in turn uses MPI to run the code on all of the other Raspberry Pis. It returns any results to the client continuously.

I created an internal website for Wee Archie, but due to the performance constraints of serving complicated websites from a Raspberry Pi while it’s handling so many communications already, I opted to make it a static website - one which does not require processing by the server other than providing the correct files. I did this using the Gatsby framework.

In order to allow the static website to start a demonstration, I wrote my own version of the Wee Archie framework in JavaScript. This framework uses the Axios library to manage communication with the demonstration server, and the React framework to provide a generic demonstration web interface.

Basic MPI Tutorials

This series of tutorials consist of a set of ten demonstrations to be run on Wee Archie and accompanying text. They are aimed at a complete beginner, who does not have programming experience, but can understand the concept of a program doing work, and take them through all of the concepts Wee MPI has to offer.

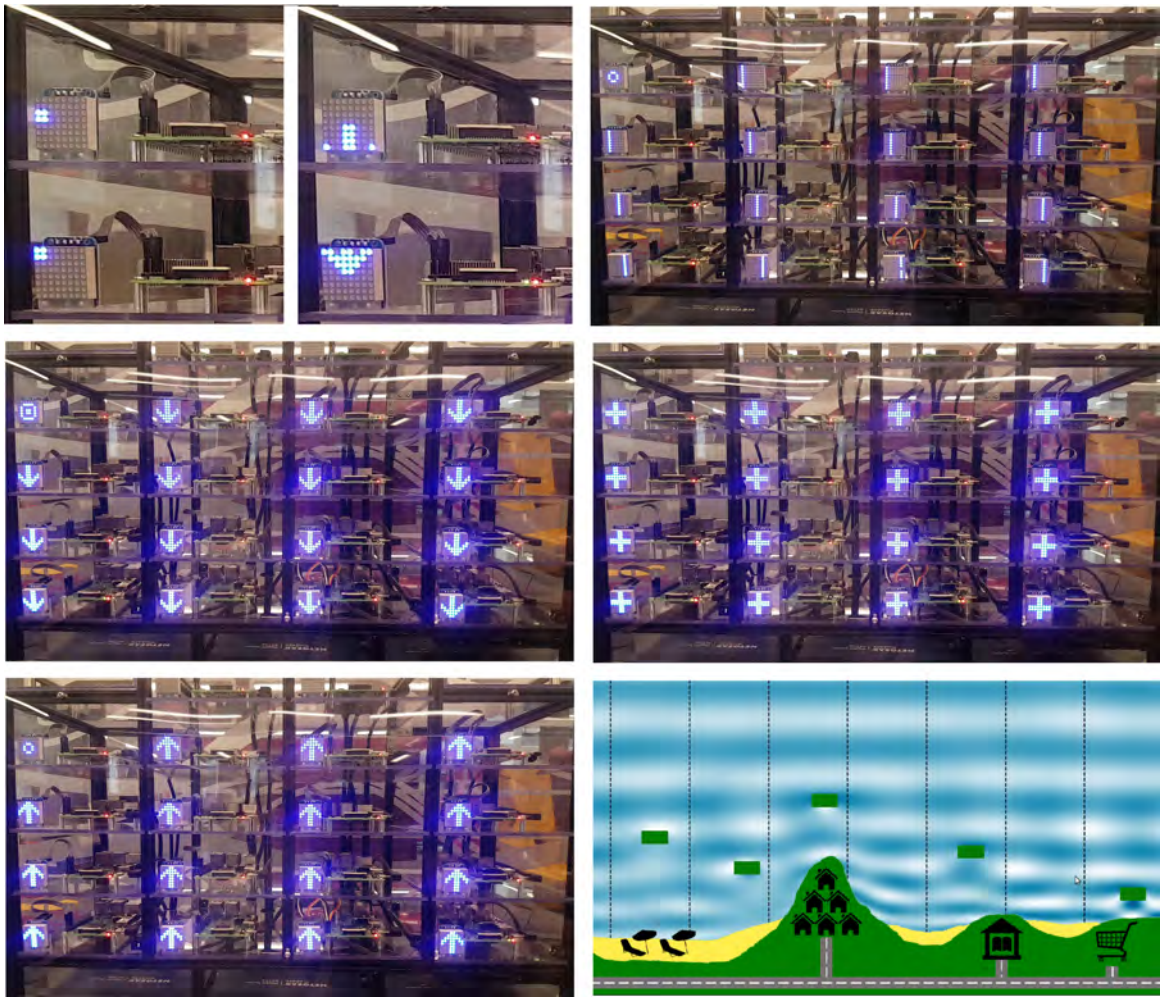


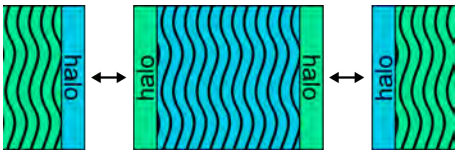
Figure 1: Left: Top: Send and Receive, Middle: Broadcast, Bottom: Gather. Right: Top: Scatter, Middle: Reduce (Sum), Bottom: Wave demonstration in progress.

The first two demonstrate the benefits of parallel computing through the analogy of cooking, showing an embarrassingly parallel problem and then, introducing conflict and making the problem less perfectly parallel, showing the need for efficient communication. The next series go through point to point communications, first showing a loop of blocking and then non-blocking sends and receives. The final set discusses collective communications, demonstrating what each do and showing their animations. It also shows the way that a broadcast could created using point to point communications.

Coastal Defence Demonstration

In the demo, the ocean is broken up into wide horizontal strips.[2] Each strip is processed by a single Raspberry Pi. However, there needs to be some communication so that waves can propagate throughout the simulation.

To do this, after every tick of the equation solver that is run in the simulation, the edges of these strips are exchanged between Pis. This is known as a "haloswap" and is shown by a custom animation. As many thousands of these occur during the simulation, I only show every hundredth haloswap.



Recommendations

The main goal of the project was to create an extensible and easily used framework to visualise parallel communications in any language. By using the animation server-client architecture and the profiling interface, this goal has been accomplished.

As demonstrated in the tutorials and coastal defence simulation, this functions in practice, and as the animations can easily be modified or turned off, there is nothing stopping adoption in future demos.

It also would be an improvement were all demonstrations for Wee Archie done using the web framework, as this would allow users to easily switch between demonstrations, and provide a surrounding explanation. It also allows for novel, interactive visualisations with the use of new features such as WebGL, and the D3 JavaScript library.

References

[1] Message Passing Interface Forum (2015). MPI: A Message-Passing Interface Standard Version 3.1

[2] EPCC, The University of Edinburgh (2018). Wee Archie Github Repository

PRACE SoHPCProject Title

Parallel Computing Demonstrators on Wee Archie

PRACE SoHPCSite

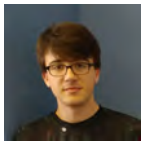
EPCC, Scotland

PRACE SoHPCAuthors

Caelen Feller, TCD, Ireland

PRACE SoHPCMentor

Gordon Gibb, EPCC, Scotland

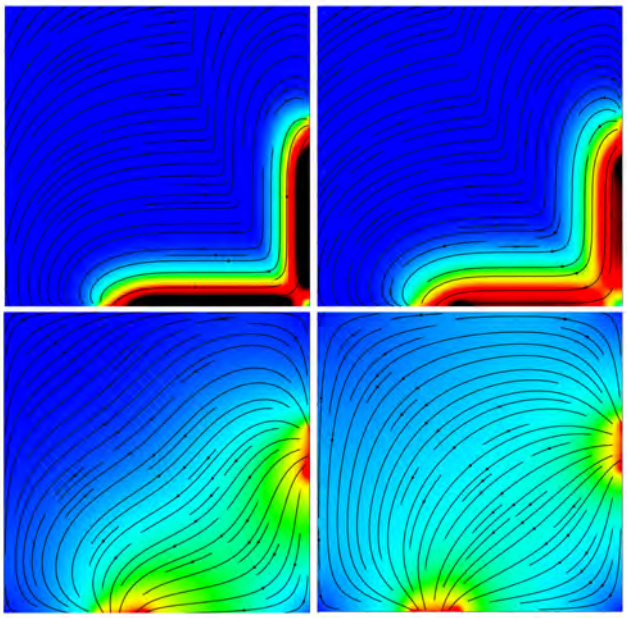


Caelen Feller

PRACE SoHPCProject ID

1907

Performance of Python Program on HPC



Ebru Diler

This project aims to improve the speed performance of Python, by experimenting with high performance scientific and fast data processing libraries.

Scientific researchers use HPC to model and simulate complex problems.They require fast supercomputers in order to resolve problems in the areas of Life Sciences, Physics, Climate research, Engineering and many others. Such machines harnesses the power of thousands of tightly connected highend servers to deliver enough power to programs. One of such machines is Archer located in Edinburgh.

In recent years, the preferred programming language for HPC is generally C/C++ or Fortran. Because they give the programmer very high control. A research or project that started with a programming language can be hard to change programming language, but there are tradeoffs that should be considered. One of other languages is Python, which is constantly growing and developed by a strong community.

Python has become a very popular programming language due to its wide range of uses. If you read programming and technology news or blog post then you might have noticed the rise of Python as many popular developer communities including StackOverflow and CodeAcademy has mentioned the rise of Python as a major programming language. Although it is widely used, it has some disadvantages. One of them is that it is poor performance. Therefore, it is not preferred in large scale modeling and simulation used in high performance computers. However, some high performance/fast processing libraries in Python also growing. Thus, tradeoff between compiled languages and scripting languages reconsidered. Python offers faster implementation time, flexibility and ease to use/learn, which makes it very strong community, which should be compared to other languages that preferly have been used in HPC.

In short, this project aims to improve the speed performance of Python, a modern programming language known with ease of use and flexibility but not with speed. Main purpose will be optimize the speed by using:

- mpi parallelization library, mpi4py
- fast data processing library, numpy

NumPy is one of the most robust and widely used Python libraries. The Python Library is a repository of script modules that can be accessed from a Python program. Recovers some frequently used commands from rewriting. It also helps to simplify programming. NumPy provides a multidimensional array object, routines for fast operations on arrays, and a range of routine products that provide a variety of derived objects (such as masked arrays and matrices), including mathematical, logical, basic linear algebra, shape manipulation, sorting, and selection.

MPI for Python provides bindings of the Message Passing Interface (MPI)[2] standard for the Python programming language, allowing any Python program to exploit multiple processors. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communications of any picklable Python object, as well as optimized communications of Python object exposing the single-segment buffer interface (NumPy arrays, builtin bytes/string/array objects).

Introduction

Computational fluid dynamics (CFD) is a branch of fluid mechanics that uses analysis methods to analyse and solve problems involving fluid flows. In order to simulate the flow of a liquid, computers are used in the calculations required. Better solutions can be achieved with high-speed supercomputers and it is necessary to solve the biggest and most complex problems with high speed supercomputers.

Fluid dynamics is a continuous problem that can be identified by partial differential equations. In this study, the finite difference approach will be used to solve the equations and determine the fluid flow model in a square cavity with a single inlet on the right side and a single outlet at the bottom of the cavity.

Computational fluid dynamics (CFD) is a branch of fluid mechanics that uses analysis methods to analyse and solve problems involving fluid flows. In order to simulate the flow of a liquid, computers are used in the calculations required.

Methods

The fluid flow can be described by the stream function defined as follows:

$$\Delta^2\psi = \frac{\delta^2\psi}{\delta x^2} + \frac{\delta^2\psi}{\delta y^2} = 0$$

Using the finite difference approach, the flow value at each grid point can be calculated as follows:

$$\psi_{i-1,j}+\psi_{i+1,j}+\psi_{i,j-1}+\psi_{i,j+1}-4\psi_{i,j} = 0$$

With the boundary values fixed, the stream function can be calculated for each point in the grid by averaging the value at that point with its four nearest neighbours. The process continues until the given number of iterations. This simple approach to solving a problem is called the Jacobi Algorithm. The velocity field u must be calculated to obtain the fluid flow pattern within the cavity. The x and y components of u are related to the stream function by:

$$u_x = \frac{\delta\psi}{\delta y} = \frac{1}{2}(\psi_{i,j+1} - \psi_{i,j-1})$$

$$u_y = -\frac{\delta\psi}{\delta x} = -\frac{1}{2}(\psi_{i+1,j} - \psi_{i-1,j})$$

Implementation

There was already C code written to solve this problem. Here are 3 different parameters that affect the algorithm:

1. The scale factor, which has to do with the size of the problem
2. The reynold number, related to the flow property of the fluid
3. Tolerance, used to provide convergence control

The flow images resulting from the change of the reynold number is given in figure 1 and 2.

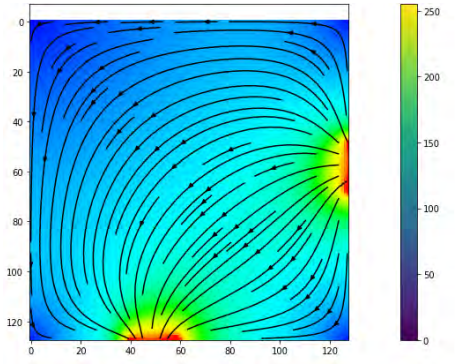


Figure 1: Fluid flow Reynold number is None

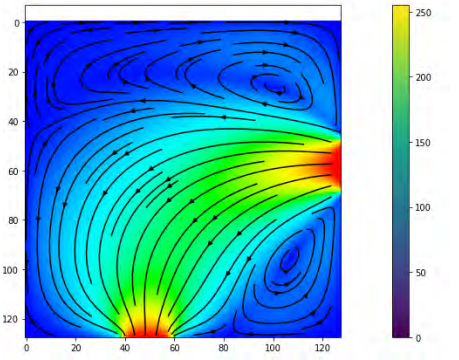


Figure 2: Fluid flow Reynold number is 2

Serial Program I have written serial code 3 different versions of Python to compare the speed.

1. With using python lists and for loops
2. With using Numpy arrays and for loops
3. With using Numpy arrays and built-in Numpy vectorization features [1]

Parallel Program I have started from a serial version running -on a single processor -and accelerated by distributing the work -over more processor cores -communicating by the MPI interface. In addition parallel approach is based on the domain decomposition. All cores calculated details and exchange necessary information due to keep the problem consistent. This is called halo swapping (Figure 3) and it's reguleted by MPI.

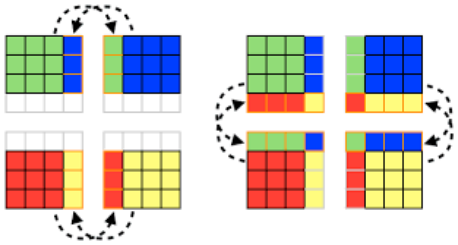


Figure 3: Visualisation of Halo Swap

Results and Discussion

Comparison of Python Programs
For each one thousand iterations, the time that spent by differents runs . The fastest program; Numpy with vectorization which let us to avoid for loops. This was our expectations at first place. But we were quite suprised with other two versions (Figure 4).

The version with Numpy worked much more slowly than the one without Numpy. Our prediction was that the program would run faster using the numpy library.

Comparison of C Programs
In order to control compilation-time and compiler memory usage, and the trade-offs between speed and space for the resulting executable, GCC provides a range of general optimization levels, numbered from 0–3, as well as individual options for specific types of optimization.

An optimization level is chosen with the command line option -OLEVEL, where LEVEL is a number from 0 to 3. The following figure shows the performance of the C program running on different optimization levels. Figure 5 shows the execution time of the C program according to the optimization level.

We also compared C and Python execution speeds. Figure 5 shows the data generated as a result of C and python programs. According to data, efficient features of Numpy has increased performance 112 times.

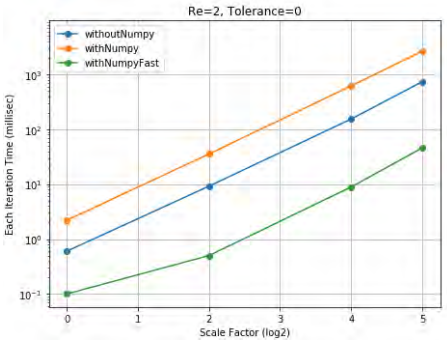


Figure 4: Comparison of Python programs

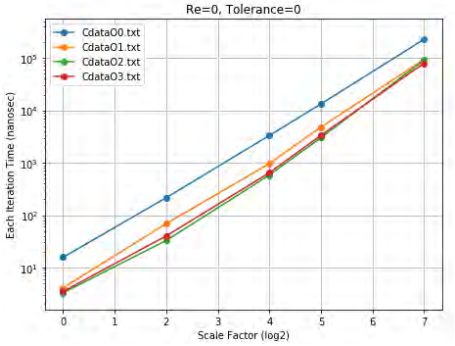


Figure 5: Comparison of C programs

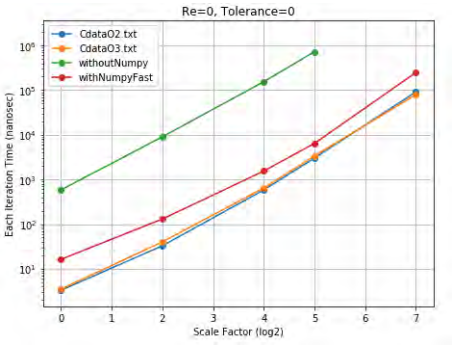


Figure 7: Comparison of Python and C programs

Performance measures for Parallelization

The speedup of a parallel code is how much faster the parallel version runs compared to a non-parallel version. Taking the time to run the code on 1 processor is T1 and to run the code on N processors is TN , the speedup S is found by:

$$S = \frac{T_1}{T_N}$$

This can be affected by many different factors, including the volume of communications to calculation. If the times are the same speedup is 1, there was no change.

Figure 7 demonstrates the strong scaling of Python parallel program running on different number of processor. The graphic describes how the execution time of the program is affected by increasing the number of cores as the size of the problem increases.

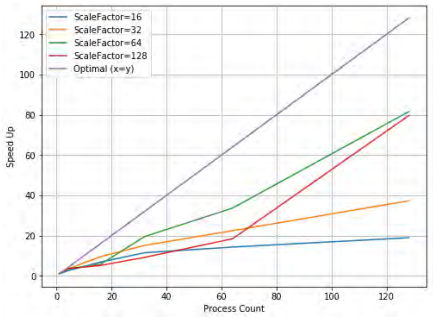


Figure 6: Parallel Python Program

Conclusion

Although the Python program still falling behind the C program in terms of performance, it is possible to improve its performance thanks to the Numpy library. In this project: we can say that using numpy arrays without using efficient numpy features (like vectorization) does effect performance in negative way for Python program.

References

[1] www.geeksforgeeks.org Vectorization-in-python

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

PRACE SoHPCProject Title

Performance of Python programs on new HPC architectures

PRACE SoHPCSite

Edinburgh Parallel Computing Centre,United Kingdom

PRACE SoHPCAuthors

Ebru Diler, [Dokuz Eylul University] Turkey

Ebru Diler

PRACE SoHPCMentor

David HENTY, EPCC, UK

PRACE SoHPCContact

Ebru, Diler, İzmir
Phone: +90 553 111 71 41
E-mail: ebrudiler0@gmail.com
linkedin.com/in/ebru-diler

PRACE SoHPCSoftware applied

Python, Numpy, mpi4py

PRACE SoHPCAcknowledgement

Great thanks to Dr. David Henty for continuous support in all matters

PRACE SoHPCProject ID

1908

Investigations on GASNet's active messages

Benjamin Huth

Illustration of the difference between the MPI approach and GASNet's active message (AM) approach. Whereas MPI requires the receiving process to actively take the data, an active message directly jump in and modify the local memory.

The project aimed at replacing an MPI-based communication backend with GASNet. However, performance tests showed that GASNet does not perform well on this task, and a following broader investigation backs this observation.

Parallelism is the most important concept in modern supercomputing. On today's large supercomputers this is mainly achieved by a technique called SPMD (Single Program Multiple Data). This means, that there is only one unique program which is executed on many different CPUs (often on several hundreds or thousands) at one time, but each of them has individual data. To communicate between these programs, programmers use specific messaging libraries. The most popular one is **MPI** (Message Passing Interface). Its basic concept is as follows:

- The sending program calls a send-function.
- The destination program has to call a receive-function.
- Only if there is a matching pair of send- and receive-functions, the communication takes place.

Although this is the most popular concept, it is not the only one. Another one called "Active Messaging" is implemented by **GASNet** (Global-Address Space Networking). Its approach is very different from the above one:

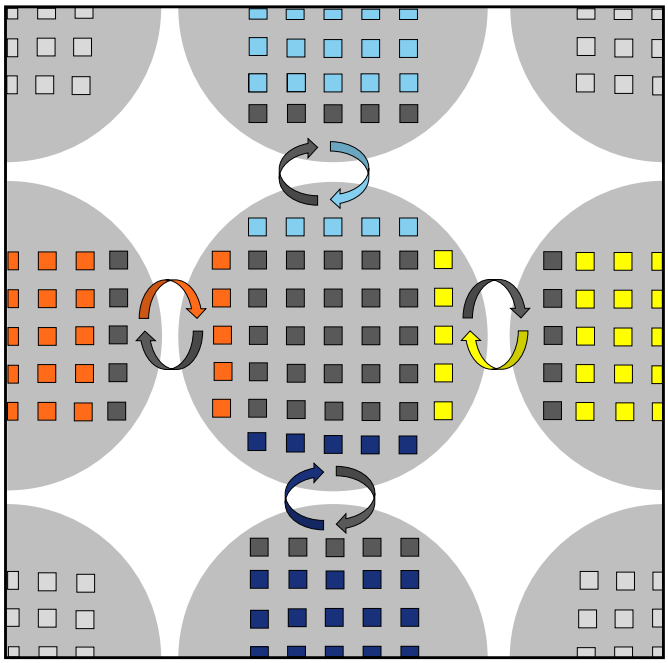
- The sender sends not just data, but also a function to the destination.
- If the message arrives at the destination, it automatically executes the function which can work on the data.
- The receiver has no direct control over the message arrivals.

There are tasks which naturally better fit to this second approach, e.g. a task-based library, which abstracts the problem of explicitly sending data from the programmer, and lets them focus on the logical structure of his program and data. Actually this was the motivation for my project: we aimed at replacing the communication back-end of a library called "EDAT", which relies on MPI, with a GASNet-based one.

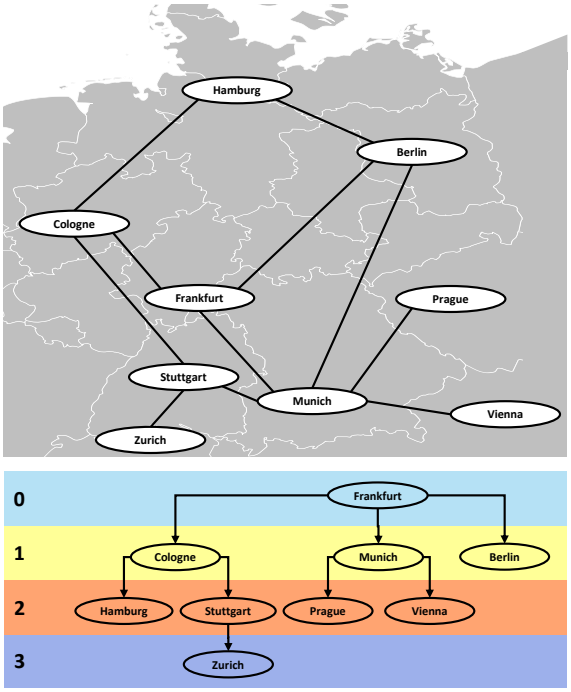
Change of focus

After implementing the first version of EDAT with GASNet we saw that the performance was not as good as expected. So we changed the focus to investigate more the basic properties of GASNet's active messaging component. To get more general results, we decided not to analyse the performance of self-made code, but to rely on existing standard benchmarks (programs for measuring hardware performance) for MPI, and rewrite them with GASNet as close as possible. These benchmarks should cover a broad spectrum of parallel software, to figure out what are the strengths and weaknesses of GASNet.

Its important to mention that we use GASNet here as a drop-in replacement for passive point-to-point communication, which is not what it is designed for.



(a) Visualisation of the swapping process in a simple 2D stencil algorithm. The grey circles represent the processes, whereas each arrow denotes a data transfer.



(b) Example for a graph. On the bottom one sees how the ”breadth-first search” goes through the above graph in four steps; In the first step it reaches all points with distance 1, in the second all with distance two...

Figure 1: Visualisation of the two benchmarks

But if that works out well, it would be easier for developer to implement than redesigning the whole application.

Benchmarks for analysis

The purpose of the first benchmark is to determine the basic messaging properties of each communicating system:

- The **bandwidth** is the amount of data a network can transfer in a certain time, nowadays modern networks typically have a low two-digit number of GB/s, e.g. 10 GB/s. Compared to the bandwidth of the internal memory this is very slow, so this is a big bottleneck for parallel software.
- The **latency** is the time a network needs to react to a messaging request. It determines the fastest possible communication, and is for small messages often more important than the bandwidth.

To measure these two quantities, we use a standard benchmark designed by the Ohio State University (see [1]).

The second application we chose for our tests is a **stencil-benchmark**. In this application a grid is distributed over many processes (so each process has a small fraction of the grid in its local memory). During the run, each grid point is updated with the sum of its 4 neighbours (in the simplest 2D case). This is a typical scientific application, so e.g. above algorithm is solving the Laplace-equation of electrodynamics. To perform an update on the grid points on the edges of a local grid, one must transfer some grid points from a neighbour process to the local one. In the simple 2D case, each process has to send 4 chunks of data to its neighbours, but also receives 4 chunks from them (see figure 1a). As a bases of the GASNet benchmark, we used the implementation of Intel's Parallel Research Kernels for MPI (see [2]).

The last benchmark is called **graph500** (see [3]). It is designed to simulate applications, which are not relying on heavy numerical computations, but on complex data structures. It implements a so called “graph”. This is an abstract data structure, which consists of points (called “vertices”), and some of them are connected by “edges”. These edges usually represent a distance, whereas the vertices can contain any kind of information. A common and simple example of data one can represent as a graph is a road map (see figure 1b).

The graph500 benchmark first creates a random graph which is again distributed over the processes, and then starts as “breadth-first search” (see also figure 1b). This results (in contrast to the stencil-benchmark) in many, but very small messages with unpredictable destinations.

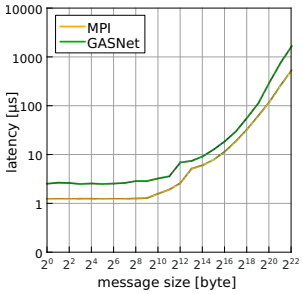
To implement this benchmark, I haven't rewritten the whole program, but just replaced the already abstracted communication layer with a GASNet based one.

All used source code is available in a repository (see [4]), together with detailed instructions how to compile GASNet and the different benchmarks.

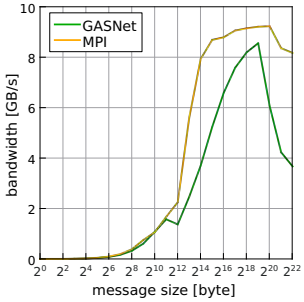
How to run the benchmarks

We run these benchmarks on three different systems, but here we show only the results from ARCHER, UK's national supercomputer. For its network there exists an optimised MPI as well as an optimised GASNet implementation, but the results are also representative for the rest of the systems.

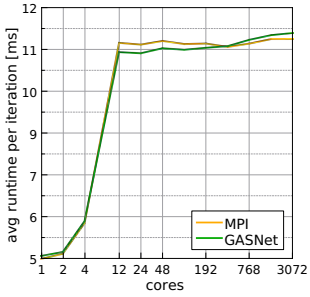
Whereas in the micro-benchmarks we scale the message-size, but hold the process count constant, for the two big benchmarks we scaled the parallelism: we started with the serial problem and 1



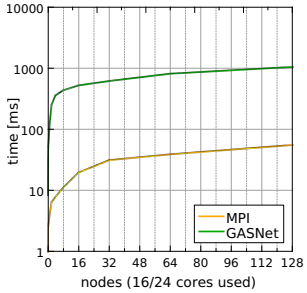
(a) Performance of the latency-benchmark. The message size is increased in powers of 2 from 1 byte to about 4 MB.



(b) Performance of the bandwidth-benchmark. The message size is increased in powers of 2 from 1 byte to about 4 MB.



(c) Performance of the stencil-benchmark. The workload is hold constant as 1000000 grid-points per process respectively core.



(d) Performance of the graph500-benchmark. The workload is hold constant as 4096 vertices per process respectively core

Figure 2: performance plots from ARCHER

process, and went up to 3072 processes (for the stencil) respectively 2048 processes (for the graph500 which requires the number of processes to be a power of two). While we are scaling the number of processes, we try to keep the amount of work per process constant (called weak scaling), since we're not interested in how the algorithm benefits from parallelism, but how well the communication works. For a theoretical communication system with infinite speed and zero latency, the execution time would stay constant, so an increase of execution time reflects the communication overhead coming with more parallelism.

Results and analysis

Already in the micro-benchmarks (see figures 2a and 2b) we see a significant performance difference. This isn't sufficient as a proof that GASNet's active messages perform badly on ARCHER, but a strong indicator.

In the latency plot, the left side is the most interesting one: Here we see that the minimal time for a message in GASNet is nearly twice as long as for MPI. On the bandwidth plot, the most interesting area is the right side, and here we see also better performance with MPI, especially for the largest message sizes.

But these effects are not directly seen in the stencil benchmark: there is not much difference between both frameworks (see figure 2c). After a big increase at the beginning (which is probably caused by memory effects inside a single node), the performance stays constant, as expected for weak scaling.

With the graph500 (see figure 2d), we see an entirely different picture. GASNet is now a whole magnitude slower than MPI. This could be surprising, because one might think, that the active messaging approach is more suitable for this kind of application (in fact, in original benchmark uses an active messaging layer implemented with MPI). On the other hand, an obvious explanation for this can be found with the latency results before: A difference in latency is much relevant with many small messages than with a few big messages.

Conclusion

With these results in mind, it is probably not worth using GASNet's active messaging as a replacement of MPI for passive point-to-point communication. For this reason, we dropped the previous aim of this project and focused on pure GASNet as an alternative for MPI. I think investigations on this topic are of general interest for the parallel computing community; although MPI is very good and widely used, it is always important to consider alternatives which can not only help developers choosing right tools for their job, but also pushing MPI to further improvements.

However, these are only preliminary results, it could be interesting to investigate a few things further, like doing more extensive profiling to understand the observed performance better. It may also be worth to investigate the possibilities of GASNet's second component RMA (Remote Memory Access) which can compete with MPI according to the developers.

References

1. Panda, D. K. *OSU Micro-Benchmarks 5.6.1* accessed 7th August 2019. Mar. 2019. http://mvapich.cse.ohio-state.edu/benchmarks/.
2. Intel Corporation. *Parallel Research Kernels* accessed 7th August 2019. 2013. https://github.com/ParRes/Kernels.
3. Graph500 Steering Committee. *Graph500-3.0.0* accessed 7th August 2019. May 2019. https://github.com/graph500/graph500.
4. Huth, B., Brown, O. T. & Brown, N. *EPCCed/GASNet-AM-benchmarks: PAW-ATM19_submission* Aug. 2019. doi:10.5281/zenodo.3368621. https://doi.org/10.5281/zenodo.3368621.

PRACE SoHPCProject Title

Task-based models on steroids: Accelerating event driven task-based programming with GASNet

PRACE SoHPCSite

EPCC, UK

PRACE SoHPCAuthors

Benjamin Huth, Uni Regensburg, Germany

PRACE SoHPCMentor

Nick Brown, EPCC, UK
Oliver Brown, EPCC, UK


Benjamin Huth

PRACE SoHPCContact

Huth, Benjamin
Universität Regensburg
Phone: +49 174 605 4097
E-mail: benjamin.huth@physik.uni-regensburg.de

PRACE SoHPCSoftware applied

MPI, GASNet

PRACE SoHPCMore Information

gasnet.lbl.gov

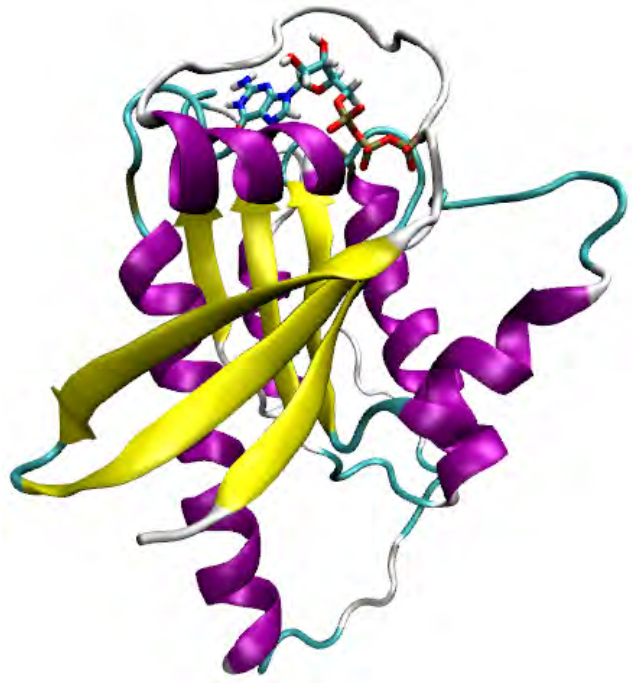
PRACE SoHPCAcknowledgement

I want to thank my mentors Nick Brown and Oliver Brown for their constant support, Ben Morse for organising our stay in Edinburgh and PRACE and Leon Kos for organising this event. Last but not least I want to thank my SoHPC collegues Caelen and Ebru for spending this awsome summer with me!

PRACE SoHPCProject ID

1909

Studying an oncogenic mutant protein with HPC



Rebecca Lait

Molecular Dynamics (MD) simulations and Normal Mode Analysis have been used in order to discover potential binding sites on the K-Ras4b protein, which could be further used in drug design studies.

The oncogenic mutant K-Ras4b is a protein that has been studied for many years but there has been limited progression regarding finding new drugs against it. Drugs are small molecules that bind to proteins and inhibit protein function. Also known as ligands, they bind to a specific region of the target protein, the binding site. In some cases, the location of the binding site is not known in advance, even though the protein structure is available. In this context, computational studies can help discover new binding sites. Hence, the goal of this project was to identify binding sites on the oncogenic protein K-Ras4b, which could benefit future therapeutic approaches.

K-Ras4b

K-Ras4b is a small GTPase, which refers to a large family of hydrolyse enzymes. It is an essential component of signalling network controlling signal transduction pathways and promoting cell proliferation and survival. It operates as a binary switch in signal transduction pathways, cycling between inactive GDP-bound and active GTP-bound states.[1] In the GTP-bound active state, cell proliferation, which refers to an increase in the number of cells due to cell growth and division, can normally occur. However, in the GDP inactive form, cell proliferation is inhibited. Figure 1 shows the GTP molecule bound to the K-Ras4b protein.

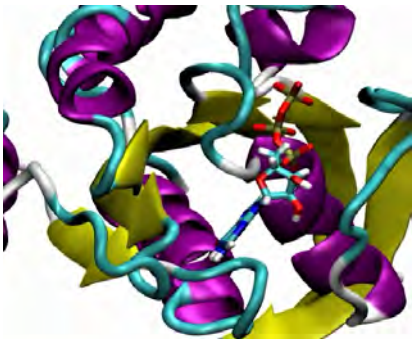


Figure 1: The GTP molecule bound to the K-Ras4b protein.

During the hydrolysis of GTP, there is a loss of a phosphate group, which results in the formation of GDP. Thus, these two states act like switches to turning the function of the protein on and off, respectively. However, the hydrolysis of GTP to GDP in the G12D mutated K-Ras4b, where the glycine in residue position 12 has been replaced with aspartic acid, is no longer possible. Consequently, the protein is 'locked' in the active GTP state.[1] This leads to overactivation of the K-Ras4b protein and as a result, to cancer growth. The K-Ras4b protein is comprised of three functional domains: the effector binding region which is defined by amino acid residues 1-86; the allosteric region by residues 87-166; and the HVR tail by residues 167-188. Each of these three domains are responsible for a particular interaction or function which contributes to the overall function of the protein.[2] The first domain in K-Ras4b, the effector binding region, is located in the N-terminus of the protein. There are three sub-domains within the effector binding region: P-loop; switch I and switch II.[1] Once the GTP is bound, the protein undergoes conformational changes that involve switch I and switch II. The second domain, the allosteric region, has

been suggested to play a role in switch II conformation and the orientation of the membrane, while the third domain, the HVR tail regulates the biological activity of the protein.[3] Within the aqueous component of the cytoplasm of a cell, the CYS185 of the protein is farnesylated, meaning that there is an addition of a farnesyl group, allowing the protein to bind to the inner leaflet of the plasma membrane.[4] There are many positively charged lysine amino acids within the HVR tail that interact strongly with the negatively charged membrane.[1] These functional domains can be seen in Figure 2. The blue region highlights the effector binding region. The red region displays the allosteric region and the green region shows the HVR tail.

Figure 2: The K-Ras4b protein with the three domains highlighted in red, blue and green.

Aims of the project

The aim of this project was to identify new potential binding sites on the surface of the unmutated wild type and the G12D mutant K-Ras4b, where small molecules can bind. This would aid the rational design of selective K-Ras4b inhibitors. In order to achieve this, Molecular Dynamics simulations and Normal Mode Analyses were performed.

Molecular Dynamics simulations

Atomic coordinates files of the protein and the specific attached ligand structure - either GTP or GDP - were downloaded from the Protein Data Bank.[5] Protein preparation including solvation and ionisation processes was carried out. The pdb files used for the unmutated WT K-Ras4b bound to GDP, G12D mutated K-Ras4b bound to GDP, unmutated WT K-Ras4b bound to GTP and G12D mutated K-Ras4b bound to GTP had pdb codes 5W22, 5US4, 5VQ2 and 6GOF, respectively.[5] Regarding the G12D K-Ras4b GTP structure, a structure was not available and so one was created using the 6GOF structure as a starting point. The HVR region was taken from pdb code 2MSC and appended to the 6GOF structure. Moreover, one of the two magnesium ions was removed and the GPPNHP molecule was replaced with the GTP molecule. The solvation step, which places the protein in a water box, and the ionization step, which places ions in water, was carried out in order to represent a more realistic biological environment. Thus, the following systems were created: unmutated wild type K-Ras4b bound to GDP or GTP, and G12D mutated
K-Ras4b bound to GDP or GTP. Once protein preparation was completed for all four K-Ras variants, Molecular Dynamics simulations were performed using the ARIS supercomputer. For the WT K-Ras4b bound to GDP, the simulation was run for 75 ns and for the remaining 3 variants for 23 ns each.

Identifying K-Ras4b binding sites

After performing Molecular Dynamics simulations, the last frame of each simulation was collected and imported into the Schrodinger Suite for binding site identification. SiteMap was used in order to determine whether any cavities with hydrophobic characteristics and hydrogen bonding capacity existed on the simulated proteins.[6] Initially, regions on the protein surface, called 'sites', were identified as potentially promising. These were located using a grid of points called 'site points'. In the second stage, contour maps, 'site maps', were generated and the predicted binding site was visualised. Finally, each site was assessed by calculating various properties, such as druggability and hydrophobicity.[6] All of the predicted binding sites were investigated and only the ones that appeared the most promising would continue in further analysis. The identified binding sites for the mutated G12D K-Ras4b bound to GTP are shown in Figure 3 below. The binding sites identified are shown in three colours: red, yellow and purple. They are each highlighted using coloured circles and are each labelled.

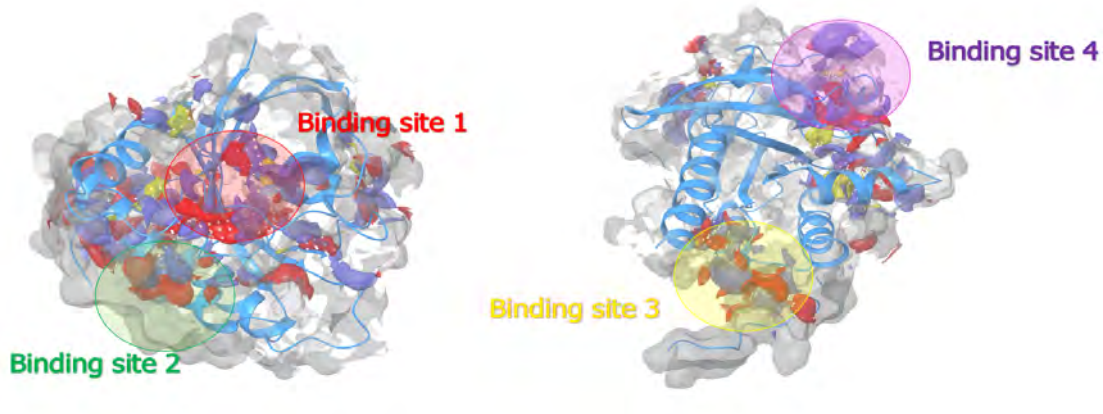


Figure 3: The identified binding sites for the G12D mutated K-Ras4b bound to GTP. Each binding site is labelled and is highlighted using coloured circles.

Normal mode analysis

Normal mode analysis was carried out alongside the binding site identification in order to investigate large scale motions of the protein structures. Normal mode vibrations are simple harmonic oscillations around an energy minimum. The normal modes which have the biggest amplitude can indicate the functional motions of the proteins. In the context of this project, the fluctuations around predicted binding sites were investigated. More specifically, if the analysis predicted large scale motions at a particular area of the protein that SiteMap identified as a potential binding site, it may be suggested that this is a biologically functional region of the protein and could act as a binding site. This type of analysis is qualitative. However, when this is combined with binding site identification results, it can give useful insights regarding the functional areas of a protein. For each of the four studied K-Ras variants, regions with considerable large-scale motions near to the identified binding sites were found. An example of this analysis undertaken is shown in Figure 4 below and compared to the binding site identification result from SiteMap.

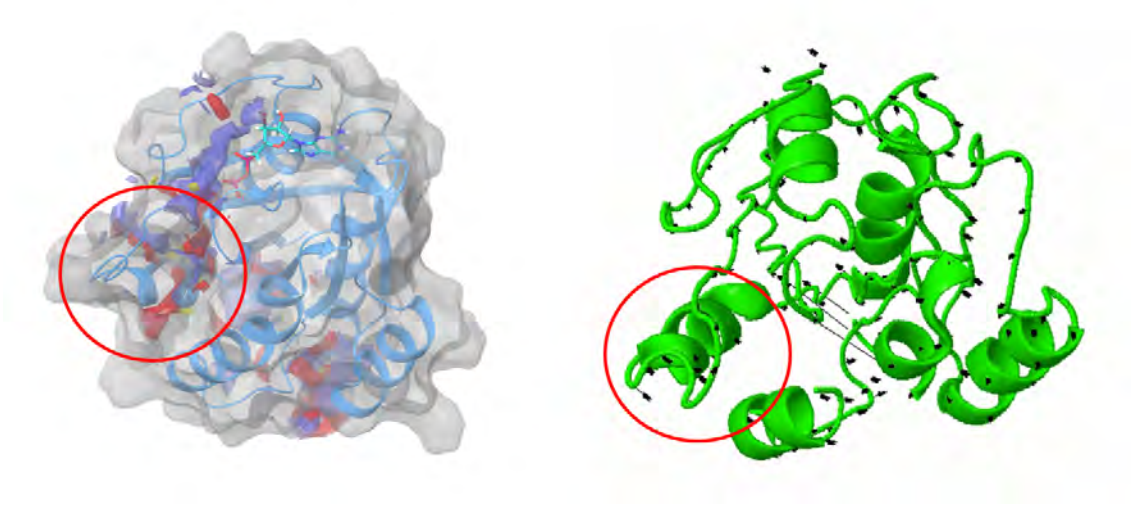


Figure 4: The normal mode analysis undertaken on the unmutated wild type K-Ras4b structure bound to GDP. The binding site is circled.

Conclusions

In this project, models of the K-Ras4b structures were constructed and MD simulations were used to study the dynamics of this protein. Binding site identification was performed in order to find cavities where candidate drugs could potentially bind, resulting in successfully identifying sites on the four K-Ras4b variants in different domains of the protein. Normal Mode Analyses were used to identify whether the predicted binding sites lie in areas of the proteins with large fluctuations.

References

[1] Lu, S., Jang, H., Gu, S., Zhang, J. and Nussinov, R. (2016). Drugging Ras GTPase: a comprehensive mechanistic and signaling structural view. Chemical Society Reviews, 45(18), pp.4929-4952.

[2] MBL-EBI Train online. (2019). What are protein domains? [online] Available at: https://www.ebi.ac.uk/training/online/course/introduction-protein-classification-ebi/protein-classification /what-are-protein-domains [Accessed 2 Aug. 2019].

[3] obbs, G., Der, C. and Rossman, K. (2016). RAS isoforms and mutations in cancer at a glance. Journal of Cell Science, 129(7), pp.1287-1292.

[4] ančík, S., Drábek, J., Radzioch, D. and Hajdúch, M. (2010). Clinical Relevance of KRAS in Human Cancers. Journal of Biomedicine and Biotechnology, 2010, pp.1-13.

[5] RCSB PDB: Homepage. [online] Rcsb.org. Available at: https://www.rcsb.org/ [Accessed 29 Aug. 2019].

[6] algren TA (2009) Identifying and characterizing binding sites and assessing druggability. J Chem Inf Model 49: 377–389.

PRACE SoHPCProject Title

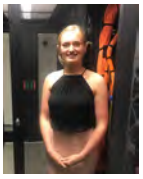
Using HPC to investigate the structure and dynamics of a K-Ras4b oncogenic mutant

PRACE SoHPCSite

Biomedical Research Foundation of Athens (BRFAA), Greece

PRACE SoHPCAuthors

Rebecca Lait, England



PRACE SoHPCMentor

Cournia Zoe, BRFAA, GREECE

PRACE SoHPCContact

Rebecca, Lait
E-mail: beckielait@googlemail.com

PRACE SoHPCSoftware applied

Virtuoso

PRACE SoHPCMore Information

www.virtouso.org

PRACE SoHPCAcknowledgement

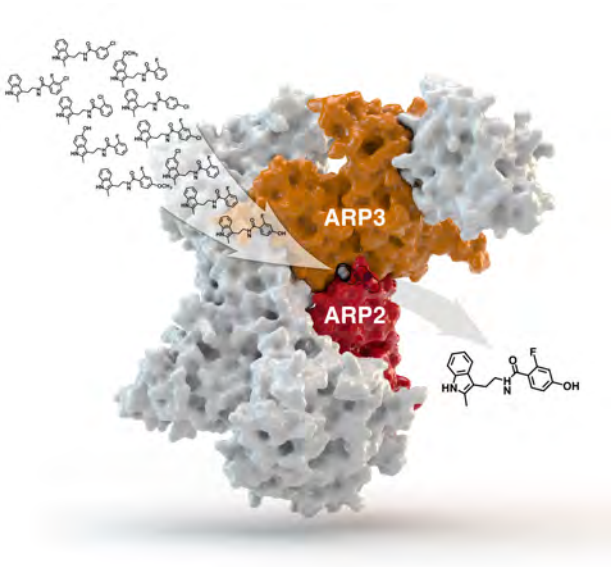
I would like to thank the PRACE Summer of HPC program for allowing me to undertake this internship. I will forever be grateful for this incredible opportunity and the experience I have gained. In addition, I would like to thank my colleagues at the Biomedical Research Foundation of Athens for their continued support and guidance. They made my project so enjoyable and taught me so many useful lessons that I know will be essential in my future career. Moreover, I am so appreciative for being granted time on the ARIS supercomputer in Athens, during this final project. Being introduced to the capabilities of this supercomputer has been inspiring. Thank you PRACE!

PRACE SoHPCProject ID

1910

Lead Optimization using HPC

Antonio Miranda



Free Energy Perturbation (FEP) calculations may reduce the time and cost of drug discovery. Specifically, they may help in optimizing the binding affinity between the drug candidate and the therapeutic target. To validate this methodology within the context of drug discovery, FEP calculations were performed and compared with experimental data. Results suggest that FEP has predictive ability and can be used for lead optimization but using HPC resources is crucial.

The cost of creating and launching one drug has been reported to be above US$1 Billion. Indeed, there are even studies that estimate this cost in more than US$2.8 billion.[1]

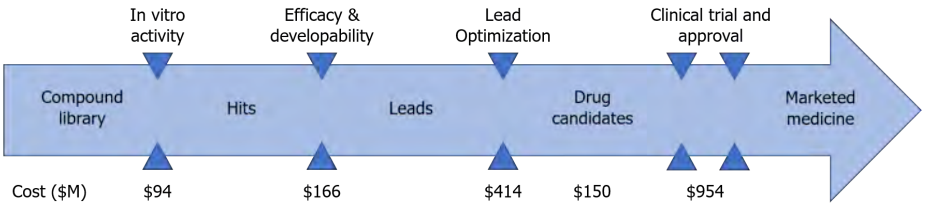


Figure 1: Drug discovery cost. Data Source: Paul et al. (2010)[7]

Drug discovery has several phases (Figure 1). One of them, "lead optimization" may account for almost 25% of the total cost of all four phases, according to a study from 2010.[1] This phase includes optimizing drug metabolism, pharmacokinetic properties, bioavailability, toxicity, and of course efficacy. Drug efficacy is directly dependent on the binding affinity of the candidate drug onto the pharmaceutical target, which is commonly a protein.

Lead Optimization using computational approaches.

This project focuses on the optimization of the binding affinity, i.e. the strength of interaction of the drug candidate onto the target protein using computational approaches. Specifically, the binding affinity of several modified compounds (analogs) of an original lead inhibitor of a target protein was studied using using Molecular Dynamics (MD) simulations coupled with Free Energy Perturbation (FEP) calculations.

Free Energy Perturbation calculations.

Free Energy Perturbation (FEP) calculations offer a promising tool for computing the relative binding affinity of two drug candidates, because they have a rigorous statistical mechanics background.[2,3] In particular, in FEP calculations the free energy difference, ΔG, between compound A and compound B is computed by an average of a function of their energy (kinetic and potential) difference by sampling for the reference state. Then, using a thermodynamic cycle (Figure 2), the relative binding free energy, $\Delta\Delta G$, between two

congeneric ligands A and B is calculated from the following equation: $\Delta\Delta G = \Delta G_B - \Delta G_A$. If this approach results in $\Delta\Delta G<0$, it is concluded that ligand B is favored over ligand A (Figure 2).

The use of FEP simulations in drug discovery has been historically restricted by the high computational demand, the limited force field accuracy and its technical challenges. However, in recent years, FEP calculations have benefited from advances in GPU and HPC availability, force-field research and the development of more elaborate simulation packages.[2]

Once these limitations have been overcome, it is time to assess whether FEP calculations using HPC resources are efficient for drug discovery. If this is the case, the time and cost of drug discovery could be reduced. This reduction could have implications in the final market drug price, which would eventually benefit the whole society. Therefore, the scope of this work is the comparison of the FEP simulation results with experimental results as well as to assess whether using HPC resources would be efficient for FEP calculations. The comparison with experimental results has been done using several analogs of CK666, a micromolar Arp2/3 inhibitor. Arp23 is a protein involved in cell locomotion and a mediator of tumor cell migration (thus, it is involved in metastasis).[4,5] The HPC resources that were used were the Greek national supercomputer, ARIS, from the GRNET facility.

Methods

For every ligand, there are two states: the ligand free in solution and the ligand bound to the protein in solution. Then, when computing $\Delta\Delta G$, there are four states: ligand A, ligand B in solution, ligand A-protein complex in solution and ligand B-protein complex in solution. In a FEP or alchemical calculations, instead of simulating the binding/unbinding processes directly (ΔG^0_1 and ΔG^0_2 from Figure 2), which would require a simulation many times the lifetime of the complex, ligand A is alchemically transmuted into ligand B in solution or in the protein environment, through intermediate, nonphysical states (ΔG^0_A and ΔG^0_B in Figure 2). Because free energy is a state function, the thermodynamic cycle of Figure 2 can be used to compute the difference in the free energy of binding between ligands A and ligands B: $\Delta\Delta G = \Delta G^0_B - \Delta G^0_A = \Delta G^0_2 - \Delta G^0_1$.

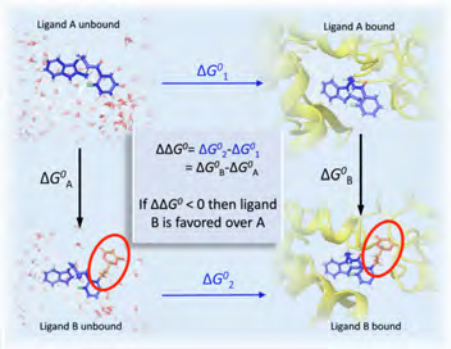


Figure 2: FEP thermodynamic cycle. Source: Athanasiou et al. (2017)[6]

In the consecutive non-physical intermediates, the interactions of the atoms are progressively transformed from the reference to the final state. In the simulations performed during this work using the software GROMACS 5.1.4,[8,9] 21 non-physical intermediates were used, split in 21 windows. This results in performing 21 FEP calculations. During the first 10 windows, van der Waals and bonding interactions were switched on for the appearing atoms. During the last 10 windows, van der Waals and bonded interactions were already switched on and electrostatic interactions were switched on progressively.

The systems need to be prepared before one can perform the FEP calculations. The setup includes preparation of the structures, the parameterization of the molecules, the alignment of the ligands, the complex formation, and the solvation and neutralization. For the protein the AMBER ff14 force field[10] was used while for the ligands we used the GAFF2 force field.[11] Ligands are aligned because it is assumed that their common core has the same binding mode. By having them aligned, there is more overlap between ligands potential energies and the FEP calculations are more probable to converge. All setup steps were performed using FESetup tool.[12] In addition, coordinate files for some of the simulated analogs were created by modifying the reference ligand with Maestro interface of Schrödinger software.[13]

After the setup, MD simulations were run on Aris supercomputer. MD simulations consist of four phases: energy minimization, two equilibration phases and production. Energy minimization is performed to relax the structures and guarantee there are no inappropriate geometries. Then, equilibration needs to be performed to stabilize the temperature and pressure of the system. Finally, each intermediate state was simulated for 5 ns during the production run each in the 21 non-physical intermediate states as described above.

At the end of the production simulation, the obtained potential and kinetic energies allow to calculate the free energy difference for the each of the intermediate steps. In this work, Bennett Acceptance Ratio, was used to compute the free energy differences.[14] The total free energy difference results by summing up the free energy differences of each intermediate step. Finally, once both legs were finished, $\Delta\Delta G$ was calculated.

Prior to performing the simulations, scaling curves were obtained in a previous phase of the project, and it was determined that complex leg calculations should be run on 4 computing nodes. The solvent leg only required 1 computing node.

Results

Figures below show the correlation among $\Delta\Delta G$ values obtained experimentally and with simulations using Gromacs and GAFF2. These correlation plots show an agreement in the values for some analogs. However, 5 out of 11 analogs retrieved results with a discrepancy larger than 1 kcal/mol. Root Mean Square Error (RMSE) was 1.42 kcal/mol, Mean Unsigned Error (MUE) was 1.13 kcal/mol.

Results obtained using Gromacs and GAFF2 force field for ligand parametrization have been compared to previous results with the same software package, using the same pipeline parameters but a different force field, GAFF. From the correlation plots, it is observed how the computed $\Delta\Delta G$ values are highly correlated with those of the previous experiments except in one case (RMSE was 0.31 kcal/mol MUE was 0.24 kcal/mol and R^2 was 0.97 ex-

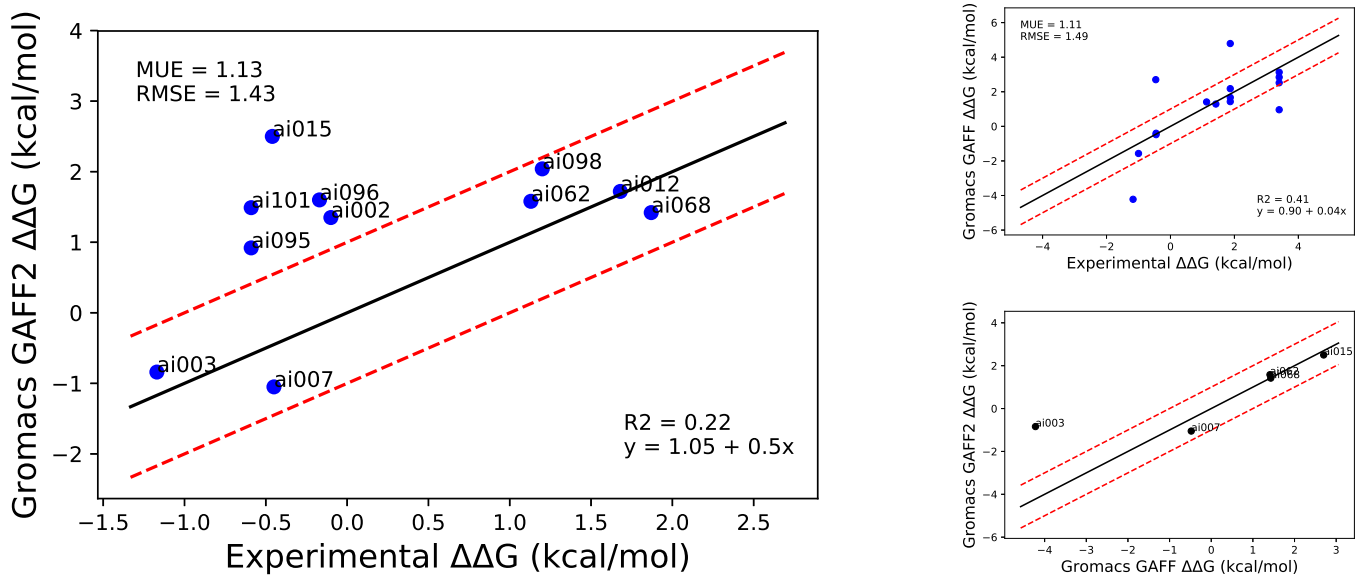


Figure 3. Left: Correlation plot between experimental and FEP results using Gromacs and GAFF2. Upper-right: correlation between experimental and FEP results with Gromacs and GAFF. Lower-right: correlation between FEP results using Gromacs and GAFF and Gromacs and GAFF2. This figure has less points because not all compounds analyzed with GAFF2 were analyzed with GAFF.

cluding that analog). For this outlier analog, the calculations using GAFF2 correlated better with experimental results.

It was previously mentioned that, every time, we were simulating 5 ns. Since there are 21 intermediate windows, the total simulated time is 105 ns per leg. For the solvent leg, every 5 ns simulation were run on 10 cores and took 6.5 hours. For the complex leg, 80 cores were used and to run every 5 ns simulation during 13.5 hours. This means that, if we were to run simulations for 100 analogs in 24 hours, we would need 84000 cores for the complex leg and 5250 cores for the solvent leg.

Discussion

Correlation plots show that, in 6 out of 11 analogs, FEP results correlate with experimental results within a 1 kcal/mol error. In general, results obtained a RMSE of 1.42 kcal/mol, MUE of 1.13 kcal/mol. This shows that FEP calculations may have predictive power and can be integrated in the drug discovery pipeline. With respect to the further refining, more work on force field parameters and on sampling techniques, the treatment of the protein-ligand environment and of chemical effects (e.g. buffer salt conditions or protonation states) may be required to further enhance FEP calculations accuracy. This study also shows that HPC resources are absolutely crucial for enabling high-throughput lead optimization with FEP.

References

[1] DiMasi, J.; Grabowski, H. and Hansen, R. Innovation in the pharmaceutical industry: New estimates of R&D costs. J. Health Eco., 2016, 47, pp.20-33.

[2] Cournia, Z.; Allen, B. and Sherman, W. Relative Binding Free Energy Calculations in Drug Discovery: Recent Advances and Practical Considerations. J. Chem. Infor. and Mod., 2017, 57(12), pp.2911-2937.

[3] Zwanzig, R. W. High-temperature equation of state by a perturbation method. I. Nonpolar gases. J. Chem. Phys. 1954, 22(8), 1420–1426.

[4] Baggett, A. W.; Cournia, Z.; Han, M. S.; Patargias, G.; Glass, A. C.; Liu, S.-Y. and Nolen, B. J. Structural Characterization and Computer-Aided Optimization of a Small-Molecule Inhibitor of the Arp2/3 Complex, a Key Regulator of the Actin Cytoskeleton, ChemMedChem, 2012, vol. 7, pp. 1286-1294.

[5] Hetrick, B.; Han, M. S.; Helgeson, L. A. and Nolen, B. J. Small Molecules CK-666 and CK-869 Inhibit Actin-Related Protein 2/3 Complex by Blocking an Activating Conformational Change," Chem and Bio., 2013, vol. 20, pp. 701-712.

[6] Athanasiou, C.; Vasilakaki, S.; Dellis, D. and Cournia, Z. Using physics-based pose predictions and free energy perturbation calculations to predict binding poses and relative binding affinities for FXR ligands in the D3R Grand Challenge 2, J Comput Aided Mol Des. 2018 Jan;32(1):21-44.

[7] Paul S.; Mytelka, D.; Dunwiddie, C.; Persinger, C.; Munos, B.; Lindborg, S. and Schacht, A. How to improve R&D productivity: the pharmaceutical industry's grand challenge, Nat Rev Drug Discov. 2010 Mar;9(3):203-14.

[8] Lindahl, E.; Hess, B.; van der Spoel, D. GROMACS 3.0: A Package for Molecular Simulation and Trajectory Analysis. J. Mol. Model. 2001, 7, 306-17.

[9] Berendsen, H. J. C.; van der Spoel, D.; van Drunen, R. A Message-Passing Parallel Molecular Dynamics Implementation. Comput. Phys. Commun. 1995, 91, 43-56.

[10] Maier, J. A.; Martinez, C.; Kasavajhala, K.; Wickstrom, L.; Hauser, K. E. and Simmerling, C. ff14SB: Improving the Accuracy of Protein Side Chain and Backbone Parameters from ff99SB. J. Chem. Theory Comput. 2015, 11, 8, 36963713.

[11] Wang, J. M.; Wolf, R. M.; Caldwell, J. W.; Kollman, P. A.; Case, D. A., Development and Testing of a General AMBER Force Field. J Comput Chem 2004, 25, 1157-1174.

[12] Loeffler, H. H.; Michel, J. and Woods, C. FESetup: Automating Setup for Alchemical Free Energy Simulations, J.. Chem. Inf. Model. 2015,55, 12, 2485-2490

[13] Schrödinger Release 2019-3: Maestro, Schrödinger, LLC, New York, NY, 2019.

[14] Bennett, C. H. Efficient estimation of free energy differences from Monte Carlo data. J. Comput. Phys. 1976, 22, 245–268.

PRACE SoHPCProject Title

HPC for candidate drug optimization using free energy perturbation calculations.

PRACE SoHPCSite

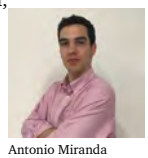
Biomedical Research Foundation of the Academy of Athens (BRFAA) and Greek Research and Technology Network (GRNET), Greece

PRACE SoHPCAuthors

Antonio Miranda, University Carlos III, Spain

PRACE SoHPCMentor

Zoe Cournia, BRFAA, Athens



Antonio Miranda

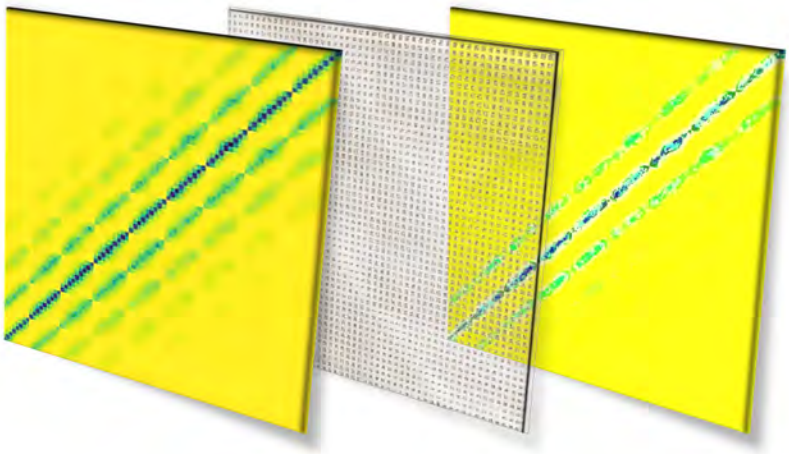
PRACE SoHPCAcknowledgement

My thanks to Dr. Zoe Cournia and the Lab members, at BRFAA, for recieving me, to Dimitris Ntekoumes for all his support and suggestions and to Dr.Dimitris Dellis at GRNET for his support and providing access to the ARIS supercomputer.

PRACE SoHPCProject ID

1911

Deep Learning for Matrix Computations



Mustafa Emre Şahin

Solving or inverting systems of linear algebraic equations(SLAE) is significant in many engineering applications and scientific fields. In this project, the implementation of stochastic gradient descent method and inverting SLAE methods will be simply explained with High Performance Computing (HPC) methods on advanced architectures.

Have you ever realised the waiting time for the machines decreasing day by day in our daily life? Of course nobody wants to wait for slow automated machines or have slow smart phones, but do you remember how long you were waiting for the old devices to open?

How to express the real-world problems to the machines?

We can describe the problems with help of the linear algebra and solve the problems by expressing as systems of linear algebraic equations. The machines understand things as numbers in a matrix, and any manipulation means matrix operations for the machines.

Figure 1: For example this is how we see images, but machines see as numbers.

”Math is boring and hard.”

Yes, everything is boring and hard when we are struggling to understand. Let’s start with a basic question:

"**Why do we need the matrix inversion?**"

Let’s assume the equation $\boldsymbol{Ax} = \boldsymbol{b}$, and we want to find **x**. To find **x**, the both sides can be divided by **A**: **x=b/A**. However, you can’t just simply divide when A is a matrix, so you need another method to solve the problem. Since $\boldsymbol{A^{-1}A} = \boldsymbol{I}$, inverse of matrix can be used to solve the problem by multiplying from left side: $\boldsymbol{x} = \boldsymbol{A^{-1}b}$. As you can see, the matrix operations are not scary monsters, they are just different.

"**What is changed in our life when we solve x?**"

X is the hero of our daily lives. It is the voltage in electronic devices, the answer of the many simulations like climate change or aerodynamics. Linear systems of equations are the foundation of the science and the technology.

The Matrix Inversion

Let’s remember the good old memories from high school, the inverse matrix can be calculated by using minors, cofactors and adjugate. However when the matrix size is large, this method is not very optimal. Iterative methods are important for the sake of rapidness while finding the inverse of the matrix.

Assume solving a system of linear equation ($\boldsymbol{Ax} = \boldsymbol{b}$ in matrix form) with $\boldsymbol{b} = \boldsymbol{I}$, by calculating $\boldsymbol{Ax} - \boldsymbol{b}$ over and over again until a solution that under desired error is obtained. Still, there are too many steps while using the iterative solvers so it can be reduced with the help of preconditioner[1].

∼ Recipe for Matrix Inverse ∼

In this project, the aim is improving non-recovered output of "Markov chain Monte Carlo matrix inversion $(\boldsymbol{MC})^2\boldsymbol{MI}$ [2]" using a stochastic gradient algorithm method. Though using the $(\boldsymbol{MC})^2\boldsymbol{MI}$ method results bad inverse matrix, inverse of matrix can be easily obtained with a time-costly recovery process [1].

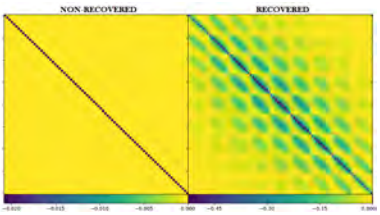


Figure 2: Outputs of $(\boldsymbol{MC})^2\boldsymbol{MI}$ method, left: Non-Recovered Inverse, right: Recovered Inverse. [1]

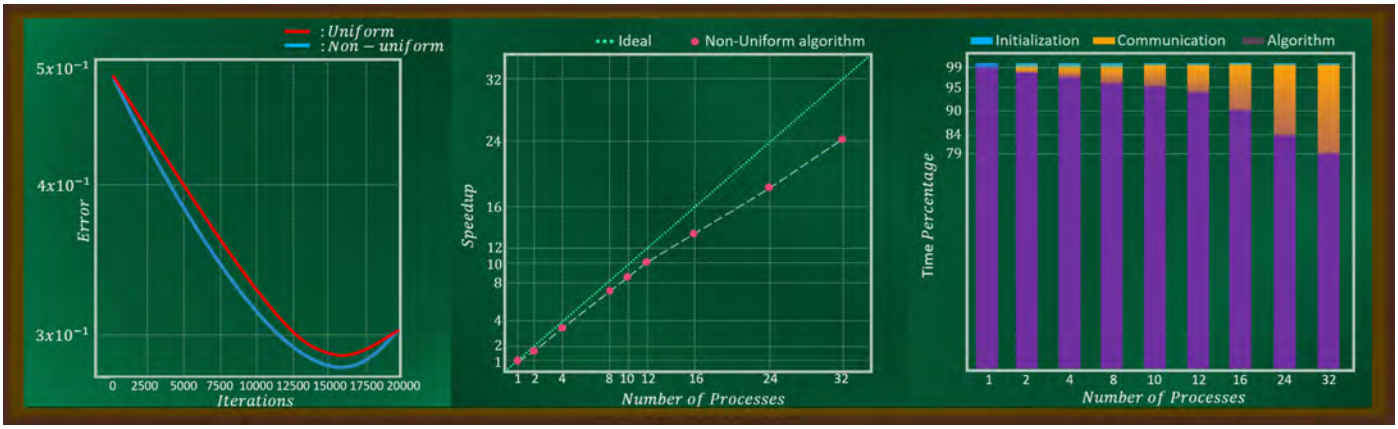


Figure 3: Left: Implementation of mSGD algorithm to non-recovered inverse of the MCMCMI. **Middle:** Speed-up graph of non-uniform mSGD with batches. **Right:** The time consumption of different parts of the code for per process.

"**What is Stochastic Gradient Descent (SGD)?**"

Gradient Descent is an iterative method which is used to minimise an objective function. If randomly selected part of the samples used in every iteration of gradient descent, it is called stochastic gradient descent.

Let's check the recipe for the method for our case.

- Assume the system $\boldsymbol{Ax = b}$ where b is Identity matrix and $\boldsymbol{x = A^{-1}}$.
- Since the aim is estimating a solution that has lower error $(Ax - b)$, the objective function should be minimised $\boldsymbol{F(x) = \frac{1}{2m}\|Ax - b\|^2}$ where $\boldsymbol{m}$ is the number of rows of A.
- After **uniform random** selection of row i for every iteration, the objective function becomes $\boldsymbol{f_i(x) = \frac{1}{m}(A_i x - b_i)^2}$.
- If we want delicious stochastic gradient descent, we should add some gradient to our objective function, $\boldsymbol{\nabla f_i(x) = \frac{1}{m}A_i{}^*(A_i x - b_i)^2}$ where $\boldsymbol{A_i{}^*}$ is the conjugate transpose of $\boldsymbol{A_i}$.
- For every round(iteration) $\boldsymbol{k}$, we should update $\boldsymbol{x}$ as $\boldsymbol{x_{k+1(new)} = x_{k(old)} - \alpha(\nabla f_i(x))}$ where $\boldsymbol{\alpha}$ is the learning rate which determines the proportion of new information that will be added.

As a deep learning method, the stochastic gradient descent algorithm in the paper [3] is implemented in Python to test our case. According to the paper for method mSGD:

- Assume a sparse matrix A which is full rank over-determined(number of rows » number of column) matrix is used on the linear system $\boldsymbol{Ax = b}$.
- Ratio of the number of non-zero elements over total number of elements in the A, p is also included to the stochastic gradient descent algorithm to estimate a better solution. With the magic of the math (Details are in the [3]), the updated x becomes:

$\boldsymbol{x_{k+1} = x_k - \alpha(\frac{1}{p^2}A_i{}^*(A_i x_k - b_i)^2 - \frac{1}{1-p^2}diag(A_i{}^* A_i)x_k)}$ where $\boldsymbol{diag()}$ gets diagonal matrix of the input.

"**Implementing mSGD to the $(MC)^2MI$**"

Although, the matrix A is a square matrix (number of rows = number of column) in our case, the method successfully **(Figure 3)** decreases the error when $\boldsymbol{MaximumSteps = 2{*}^4, \alpha = 4 * 10^{-4}}$ and $\boldsymbol{A^{-1}}$, non-recovered inverse from $\boldsymbol{(MC)^2MI}$ given as initial solution $\boldsymbol{(x)}$ to the system. After successful results in Python, the algorithm is implemented in **C++ with Eigen library on Scafell Pike X1000 Supercomputer**.

"The Cherry on the cake"

"**Life is not fair. Why should random selection be fair?**"

In mSGD, the rows are selected with uniform probability. The results are slightly better (Figure 3) if probability of selecting a row is proportional to the norm of the row.

"**Last touch: Adding Batches to Parallelization**"

Stochastic Gradient Descent and Batching are as like as two peas in a pod. Instead of using whole matrix A for the method,rows are divided into the subsets for each process in the parallelization with Hybrid MPI/OpenMP.When rows are not equally divided, it is a good trick to give lower amount of rows to the master process since it has more work than others.

Discussion and Conclusion

I successfully implemented and parallelized Stochastic Gradient Descent method to the $\boldsymbol{(MC)^2MI}$ algorithm with the non-uniform random distribution, the batching and a successful speed-up. As the number of processes increase, the communication took more time than others, however speed-up is better as shown in the Figure 3's Middle and Right images. The further acceleration methods for the algorithm can be investigated by future generations of Summer HPC participants.

References

[1] Lebedev, A. (2017). Accelerating stochastic methods for the SLAE. SoHPC Final Reports.

[2] Alexandrov, V. N. (1998). Efficient parallel Monte Carlo methods for matrix computations.

[3] Ma, A., Needell, D. (2017). Stochastic Gradient Descent for Linear Systems with Missing Data.arXiv:1702.07098.

PRACE SoHPCProject Title

Hybrid Monte Carlo/Deep Learning Methods for Matrix Computation on Advanced Architectures

PRACE SoHPCSite

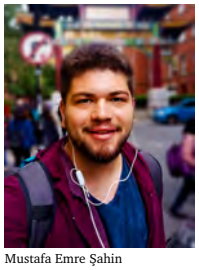
STFC Hartree Centre, United Kingdom

PRACE SoHPCAuthors

Mustafa Emre Şahin,
İzmir Institute of Technology, Turkey

PRACE SoHPCMentor

Prof. Vassil Alexandrov,
STFC Hartree Centre, UK



Mustafa Emre Şahin

PRACE SoHPCAcknowledgement

I am deeply thankful to my project mentor Prof.Dr.Vassil Alexandrov and my shifu Anton Lebedev. Many thanks to Assoc.Prof.Dr. Leon Kos for his helps during problematic times. I'm grateful to my mentors Assist.Prof.Dr. Işıl Öz and Prof.Dr. Ferit Acar Savacı for helping me to build my personality and career.

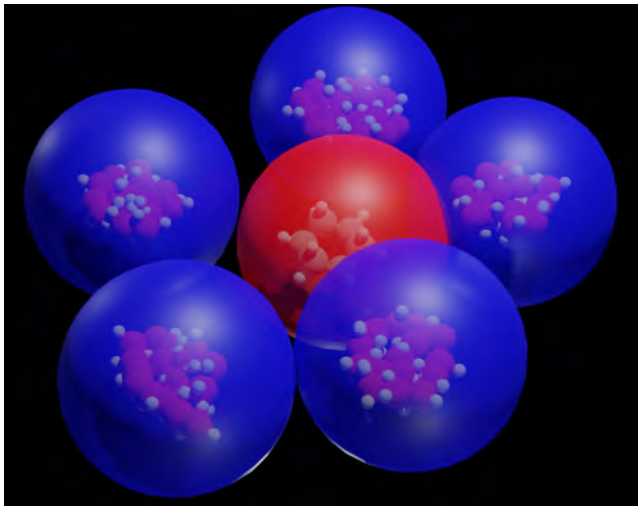
PRACE SoHPCProject ID

1912

Performance evaluation of a dissipative particle dynamics code for very large systems using latest hybrid CPU-GPU architectures.

Billion bead baby

Davide Di Giusto



Dissipative particle dynamics simulations allow to overcome the length-scale limits of molecular dynamics, while maintaining the inner, discrete nature of matter. To simulate realistic applications, a huge system is necessary, where the number of particles is proportional to some billions. This project aim is to extend the current size limits of the DL-meso DPD code and scale it on a large GP-GPUs architecture.

The Dissipative Particle Dynamics (**DPD**) method provides a powerful tool to simulate physical phenomena of common interest. The approach implies the definition of a particle size, which can include several small molecules or few more complex ones. The action of three forces, conservative, dissipative and stochastic ones, allows this method to be valid for the simulation of simple and complex fluids. Finally, the DPD deals with the mesoscale, between the nanoscale, that describes the size of the molecules, and the microscale, that starts to measure the continuum media. Potentially, large scale simulations based on the DPD can try and overcome the distance that separates the mesoscale and the microscale, while still mantaining features derived from the particle size, closer to the nanoscale of matter: this could benefit both research and industrial worlds. The DL Meso DPD code (DL Meso) implements the Dissipative Particle Dynamics method and, thanks to the usage of GPU accelerators, we will explore its size limits and extend them, in order to perform large scale simulations.

The DL Meso DPD code is written in Fortran but offers the possibility to transfer the calculations on GPUs (Graphics Processing Unit), as the main loop has been ported to CUDA C. Let us consider the original version of the code. Aim of the project was to target its size limitations, so to extend the simulation possibilities. From previous tests, the largest number of beads that had been deployed was **two billions**. Not bad at all, we could say! But we should be ambitious in our work and tried performing a three billion simulation. For it to run, several quantities needed to be stored on the machine memory in the corresponding variables, defined by a size and a type. Moreover, many of these were directly proportional to the number of beads stocked for the simulation, therefore we assumed straightforwardly that such parameter was causing integer overflow for the mentioned variables. In computer programming, overflow occurs when an integer value is stored on a variable whose memory allocation is not big enough for the current necessity. For example, if a integer value exceeds the typical value range of a 4 bytes integer, which goes from -2147483648 to 2147483647, the code is trying to improperly access the system memory, leading to its termination: for sure this was happening for the variable that stocked the total number of beads in the system (3 billions), but sneakier ones were hiding in the scripts.

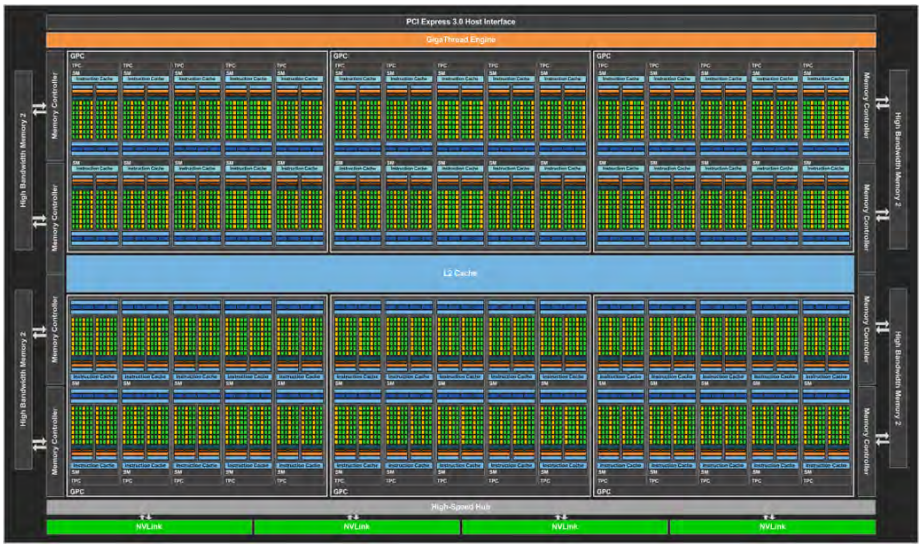


Figure 1: NVIDIA Tesla P100 representation: it is possible to observe the moltitude NVIDIA CUDA cores, organised in streaming multiprocessors. The huge number of cores determines its inner parallelism.(Source: Link)

The complexity of this problem, together with the articulate structure of the code, required proper tools for debugging. The Eclipse™ integrated development environment and the Nsight™ Eclipse Edition were chosen, as they allow a visual exploration of the code lines while running in debug mode: in this way, it was possible to understand which variables needed to be changed and which ones could remain of the same integer type. Once the code was ready to go, we could start the strong scaling experiment. Basically, consecutive runs of the same code execution were performed: the first was the serial one, which set a reference time that was required for its completion. Then, the parallel executions were performed, doubling the computational resources each time. This was possible because the DL Meso DPD code is prone to parallel executions, following the typical domain decomposition implemented using MPI libraries: in essence, the system volume is split equally between the processes, while the particles jump from one to the other as they move.

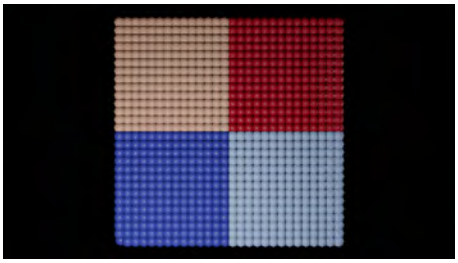


Figure 2: Typical domain decomposition for 4 parallel processes: each colour indicates a different process identification number.

The comparison between the execution times returns the **speedup** in function of the number of parallel executions. When the speedup is ideal, meaning that for twice the processes our execution will take half the time, the code is said to scale well; but the ideal scaling is undermined by the communication between parallel processes. As mentioned before, the particles move through the system volume, while the parallel decomposition scheme remains static as the simulation progresses in time. Therefore, every time-step messages between all the processes need to be sent, in order to redistribute the beads according to their most recent position.

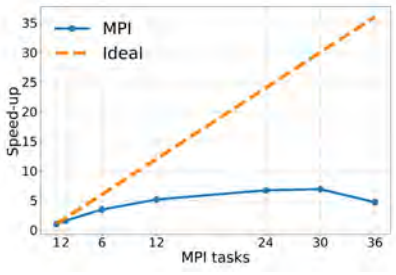


Figure 3: DL Meso code scaling for MPI only version. For a small number of particles the system does not scale well.

Obviously, the larger the number of parallel executions, the larger the total amount of **communication** required. Eventually, while scaling our code, the communication time will be comparable with the time spent for the calculations, resulting in a deviation from the ideal speedup. At this point, we just needed a proper computer to work with. Our basic necessity was a large GPU architecture. Luckily, we could obtain access to Piz Daint: this is the largest supercomputer in Europe, currently 6th in the top 500 list (Top 500 list), managed by the Swiss National Supercomputing Centre (CSCS Piz Daint). Moreover, Piz Daint offers 5704 hybrid nodes, equipped with one 12 cores CPU and one NVIDIA®Tesla®P100 each. These are indeed terrific numbers, but we probably need to better define the difference between CPU and GPU.

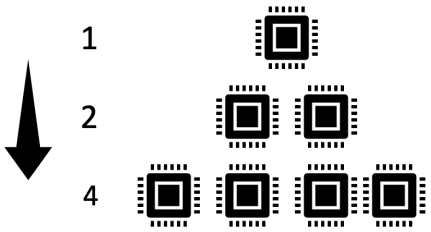


Figure 5: Representation of the running configuration: each electronic circuit consists of one single task CPU-GPU hybrid node.

Whereas a CPU is the general electronic circuit which runs a computer and can be multicore, presenting already a certain attitude towards parallelism, a **GPU** is a specialised device, highly efficient in large data block manipulation. Originally GPUs were created to accelerate image processing, but eventually their potential in High Performance Computing was clear. Since then, they have been deployed as **accelerators** in the majority of the supercomputers.

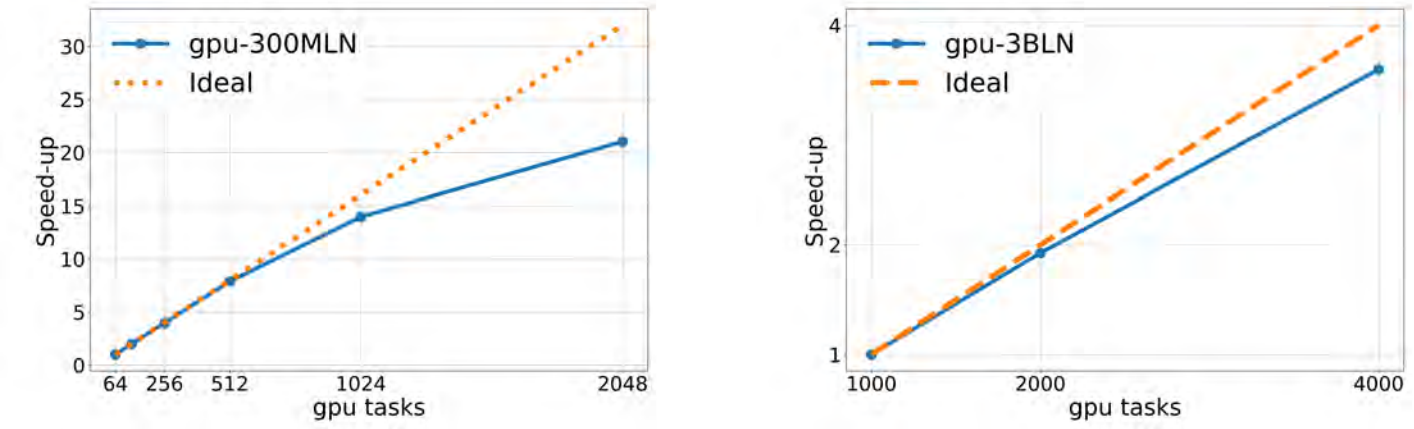


Figure 4: Dl Meso DPD code scaling on hybrid CPU-GPU nodes for a total number of particles of: left) 300 millions; right) 3 billions.

Table 1: CPU vs GPU: acceleration

CPU Time elapsed		
Nodes	CPU	GPU
1	3401.98	195.82
2	2264.75	147.13
16	334.06	75.22
32	167.28	36.27

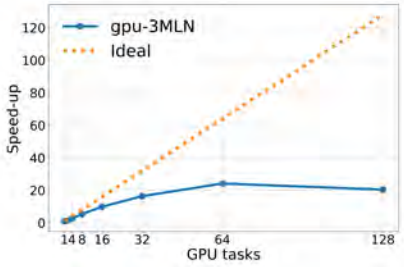


Figure 6: Dl Meso DPD code scaling on hybrid CPU-GPU nodes for 3 million particles

In table 1 is possible to appreciate the acceleration granted when using the GPUs. Our running configuration was the following: we wanted one GPU per process, so we would demand only single task CPU-GPU hybrid nodes. In figures 4 and 6 the scaling of the Dl Meso DPD code is plotted, for system sizes of 3 millions in figure 6, and 300 millions and 3 billions in figure 4, respectively left and right. We can notice that the reference case is not always the serial version. This craftiness is necessary, as the GPU memory is limited to 16 Gb, whereas the size of the arrays allocated during execution can be larger. The obtained speedups show that the code scales well for large numbers of beads. The 3 billion case scales almost ideally up to 4000 nodes, meaning this kind of big system simulations can be accelerated efficiently. The result for the 300 million case shows good scaling too, providing also an indication for a match between job size and number of parallel tasks. Instead, when using only 3 million particles, the code does not scale well. In conclusion, we have acknowledged that GPU accelerators are a great opportunity to benefit high performance computing, as they can outclass the GPU-less applications. Anyway, even on the hybrid CPU-GPU nodes the code could scale non-ideally. This is not straightforward: the GPUs are voracious in terms of total number of particles, as they require large amounts, but grant brilliant speedup. Eventually, the larger the number of particles stocked, the larger the number of nodes whose usage leads to good scaling: this is really meaningful as it confirms the possibility of efficiently accelerating large scale simulations based on the Dissipative Particle Dynamics method. Moreover, the scaling possibilities are limited on the minimum number of nodes that are required in order to run the first experiment. Clearly, this limitation is set by the GPUs, as they posses a 16 Gb memory, pretty low when compared to the CPU one, that counts up to 64 Gb per node on Piz Daint.

The results we obtained during these two months suggest the possibility of efficiently simulating large system size DPD simulations. This outcome can eventually benefit both the industrial world, as more realistic experiments could be performed on the computers, reducing costs and environmental footprint, and the research one, opening new perspectives of study. Future developments could try performing simulations for even larger number of particles. Having already tried running simulations for 6 billion particles, the communication between processes can become critical and lead to failure, therefore this should kept into account. Moreover, some other features could be scaled, like the Fast Fourier Transformate algorithm, which can be tricky on large hybrid architectures.

I would like to specially thank CSCS for the preparatory access (request 2010PA5022) that allowed me to use Piz Daint and for the support during the whole summer.

References

[1] M. A. Seaton, R. L. Anderson, S. Metz and W. Smith, (2013). DL-MESO: highly scalable mesoscale simulations, Mol. Sim. 39 (10) 796-821

PRACE SoHPCProject Title

Scaling the Dissipative Particle Dynamic (DPD) code, DL MESO, on large multi-GPGPUs architectures

PRACE SoHPCSite

STFC Hartree, United Kingdom

PRACE SoHPCContact

Davide Di Giusto, [University of Udine] Italy

Davide Di Giusto

PRACE SoHPCMentor

Jony Castagna, STFC Hartree, U.K.

PRACE SoHPCContact

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

PRACE SoHPCSoftware applied

Eclipse®(IDE), Nvidia Nsight®

PRACE SoHPCMore Information

Nvidia developer blog

PRACE SoHPCAcknowledgement

I would like to thank Leon Kos for the guidance that he has granted all summer long; moreover, I am really thankful to all the people at Prace for the organisation of the SoHPC. Then, I have nothing but gratitude for my mentor Jony Castagna, who is somebody to look up to and managed to teach me so much in little time. Finally, I would like to thank Nia for the organisation, Vassil Alexandrov and Luke Mason and all the people at the Hartree centre for the continuous support and kindness.

PRACE SoHPCProject ID

1913

Investigating possibilities to accelerate applications deployed on the edge that utilize Deep Neural Networks.

Switching up Neural Networks

Igor Kunjavskij

Using the Intel Movidius Neural Compute Stick, I researched on the possibilities of switching models during runtime.

Smart devices permeate the fabric of our daily lives, be it the phone in everybody's pocket or the smart watches some of us wear. But not only our daily lives are filled with these devices, they also start to appear in areas which used to be inaccessible for humans. This could be a device attached to satellites in space running earth observation missions. Or underwater robots conducting maintenance on critical structures such as underwater pipelines. In these areas, small devices tend to take on more and more compute intensive tasks such as processing images that they capture. But especially computer vision and processing images encompass in general very compute intensive tasks, even more so if Deep Neural Networks are involved. Lightweight devices that do not provide too much compute power, as for example the Raspberry Pi, can quickly become overwhelmed with these tasks. A solution to this problem would be to just send the data that a device collects to a data center. But given the fact that these devices are deployed in regions with restricted connectivity, sending data back and forth for processing purposes is not a viable option. That's where the Intel Movidius Neural Compute Stick comes into play, an accelerator in USB format that alleviates a lot of those problems, making it possible for images to be processed on the edge on the device itself.

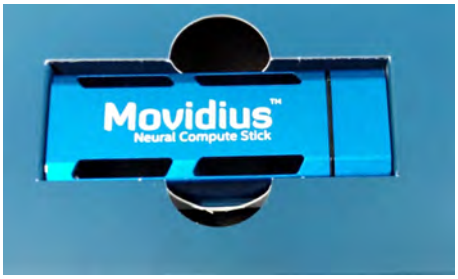


Figure 1: The Intel Movidius Neural Compute Stick.

Intel Movidius Neural Compute Stick

The Intel Movidius Neural Compute Stick was developed specifically for computer vision applications on the edge, with dedicated visual processing units at its disposal. Once a Deep Neural Network was trained on powerful hardware for hours or sometime days, it needs to be deployed somewhere in a production environment to fulfill its purpose. This can be for example in computer vision applications on the edge, which can encompass detecting pedestrians or other objects in images or also classifying objects in detected images such as different road signs. The deployment of such an application can happen on a server in a data center, an on-premise computer or an edge device. Now the crux with the latter is, that applications such as computer vision tend to have quite a high computational load on hardware, whereas edge devices tend to be lightweight and battery powered platforms. It is of course possible to circumvent this problem by streaming the data that an edge device captures to a data center and process it there. But the next issue that the term "edge device" already implies is, that these computing platforms are situated in quite inaccessible regions, like on an oil rig in the ocean or attached to a satellite in outer space. Transmitting data is in these cases costly and comes with a lot of latency, rendering real time decision making nearly impossible. Especially if this data encompasses images, which as a rule of thumb tend to be large. That's where hardware accelerators like the Neural Compute Stick bring in a lot of value. Instead of sending images or video to some server for further analysis, the processing of data can happen on the device itself and only the result i.e. pure text is sent for storage or statistical and visualization purposes

to some remote location. Such an approach brings numerous benefits, but most importantly alleviates latency and communication bandwidth related concerns. Now what is this Neural Compute Stick all about? The Intel Movidius Neural Compute Stick is a tiny fan device that can be used to deploy Artificial Neural Networks on the Edge. It is powered by an Intel Movidius Vision Processing Unit, which comes equipped with 12 so called "SHAVE" (Streaming Hybrid Architecture Vector Engine) processors. These can be thought of as a set of scalable independent accelerators, each with their own local memory, which allows for a high level of paralleled processing of input data.

OpenVino

Complementing this stick is the so called Open Visual Inferencing and Neural Network Optimization (OpenVino) toolkit, a software development kit which enables the development and deployment of applications on the Neural Compute Stick. This toolkit basically abstracts away the hardware that an application will run on and acts as an "mediator". To make use of the Neural Compute Stick, it is necessary to first install the Open Visual Inferencing and Neural Network Optimization (OpenVino) toolkit, which I briefly introduced in my last post. This toolkit aims at speeding up the deployment of neural networks for visual applications across different platforms, using a uniform API. Training a Deep Neural Network is like taking a "black box" with a lot of switches and then tuning those for as long as it takes for this "black box" to produce acceptable answers. During this process, such a network is fed with millions of data points. Using these data points, the switches of a network are adjusted systematically so that it gets as close as possible to the answers we expected. Now this process is computationally very expensive, as data has to be passed through the network millions of times. For such tasks GPUs perform very well and are the de facto standard if it comes to hardware being used to train large neural networks, especially for tasks such as computer vision. If it comes to the frameworks used to train Deep Neural Networks, so there are a lot that can be utilized like Tensorflow, PyTorch, MxNet or Caffe. All of these frameworks yield a trained network in their own inherent file format. After the training phase is completed, a network is ready to be used on yet unseen data to provide answers for the task it was trained for. Using a network to provide answers in a production environment is referred to as inference. Now in its simplest form, an application will just feed a network with data and wait for it to output the results. However, while doing so there are many steps that can be optimized, which is what so called inference engines do. Now where does the OpenVino toolkit fit in concerning both tasks of training and deploying a model? The issue is that using algorithms like Deep Learning is not only computationally expensive during the training phase, but also upon deployment of a trained model in a production environment. Eventually, the hardware on which an application utilizing Deep Learning is running on is of crucial importance.

Neural Networks, especially those used for visual applications, are usually trained on GPUs. However, using a GPU for inference in the field is very expensive and doesn't pay off when using it in combination with an inexpensive edge device. It is definitely not a viable option to use a GPU that might cost a few hundred dollars to make a surveillance camera or a drone. If it comes to using Deep Neural Networks in the field, most of them are actually running on CPUs. Now there are a lot of platforms out there using different CPU architectures, which of course adds on to the complexity of the task to develop an application that runs on a variety of these platforms.

That's where OpenVino comes into play, it solves the problem of providing a unified framework for development which abstracts away all of this complexity. All in all, OpenVino enables applications utilizing Neural Networks to run inference on a heterogeneous set of processor architectures. The OpenVino toolkit can be broken down into two major components, the "Model Optimizer" and the "Inference Engine". The former takes care of the transformation step to produce an optimized Intermediate Representation of a model, which is hardware agnostic and useable by the Inference Engine. This implies that the transformation step is independent of the future hardware that the model has to run on, but solely depends on the model to be transformed. Many pre-trained models contain layers that are important for the training process, such as dropout layers. These layers are useless during inference and might increase the inference time. In most cases, these layers can be automatically removed from the resulting Intermediate Representation. Even more so, if a group of layers can be represented as one mathematical operation, and thus as a single layer, the Model Optimizer recognizes such patterns and replaces these layers with just one. The result is an Intermediate Representation that has fewer layers than the original model, which decreases the inference time. This Intermediate Representation comes in the form of an XML file containing the model architecture and a binary file containing the model's weights and biases.

After using the Model Optimizer to create an Intermediate Representation, the next step is to use this representation in combination with the Inference Engine to produce results. Now this Inference Engine is broadly speaking a set of utilities that allow to run Deep Neural Networks on different processor architectures. This way a developer is capable to deploy an application on a whole host of different platforms, while using a uniform API. This is made possible by so called "plugins". These are software components that contain complete implementations for the inference engine to be used on a particular Intel device, be it a CPU, FPGA or a VPU as in the case of the Neural Compute Stick. Eventually, these plugins take care of translating calls to the uniform API of OpenVino, which are platform independent, into hardware specific instructions. This API encompasses capabilities to read in the network's intermediate representation, manipulate network information and most importantly pass inputs to the network once it is loaded onto the target device and get outputs back again. Now a common workflow to use the inference engine includes the following steps:

1. Read the Intermediate Representation - Read an Intermediate Representation file into an application which represents the network in the host memory. This host can be a Raspberry Pi or any other edge or computing device running an application utilizing the inference engine from OpenVino.
2. Prepare inputs and outputs format - After loading the network, specify input and output precision

and the layout of the network.

3. Create Inference Engine Core object – This object allows an application to work with different devices and manages the plugins needed to communicate with the target device.
4. Compile and Load Network to device – In this step a network is compiled and loaded to the target device.
5. Set input data - With the network loaded on to the device, it is now ready to run inference. Now the application running on the host can send an "infer request" to the network in which it signals the memory locations for the inputs and outputs.
6. Execute - With the input and output memory locations now defined, there are two execution modes to choose from:
   (a) Synchronously to block until an inference request is completed.
   (b) Asynchronously to check the status of the inference request while continuing with other computations.
7. Get the output – After the inference is completed, get the output memory.

Another problem that remains though, even with the Neural Compute Stick accelerating computations, is that once a model is deployed on an edge device, it's hard to repurpose the device to run a different kind of model. This is where the "dynamic" part of the title of my project comes into play. The Neural Compute Stick is a highly parallelized piece of hardware, with twelve processing units independently running computations. This way, it is possible to not only have one, but several models loaded into its memory. Even more so, it is possible to switch these models and load new models into memory. This allows to adapt to new situations in the field, like when the feature space changes or the things that are supposed to be detected change. The simplest case of such an occurrence might be the sun setting or bad weather conditions coming up. Another motivation to switch models might also be to save power, as edge devices tend to have limited capacities if it comes to energy sources. Instead of deploying one big model that is supposed to cover all cases that could occur in a production environment, it would be possible to have many small models that could be loaded in and out of memory at runtime.

Model switching prototype

In my project I investigated the feasibility of doing so and implemented a small prototype that switches models at runtime. This prototype switches between two models detecting human bodies and faces. These models are both so called Single shot detector MobileNets, networks that are better suited to be deployed on smaller devices which localize and classify an object in a single pass through the network drawing bounding boxes around it. I used OpenCV for this task, which is a library featuring all sorts of algorithms for image processing. Next to OpenCV I had OpenVino running as a backend, a toolkit accompanying the Neural Compute Stick, which allows to utilize it in an optimized manner. I eventually tested this model switching prototype by loading and offloading models in and out of memory of the Neural Compute Stick. I did this with a very high frequency of one switch per frame to determine what the latency of such a model switch would be in a worst-case scenario. The switching process includes reading the input and output dimensions of a model by using the XML representation of its architecture and then loading it into the memory of the Neural Compute Stick. On average this switch caused an extra overhead of about 14 percent of the overall runtime. To put this into absolute numbers, on average it took my application half a second to capture and generate an output for an image, whereas a model switch in between would add a little less than a tenth of a second to this time.

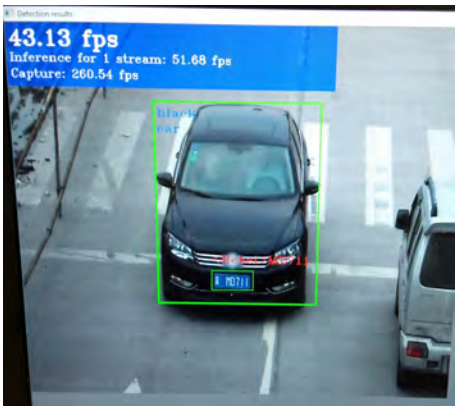


Figure 2: Example of using a SSD MobileNet in combination with the Neural Compute Stick.

Of course, there is a lot of room for improvement given these numbers. One such improvement would be concerned with the parsing of the model dimensions. I used a simple XML parser to do so and had to read in the input and output dimensions of a model on every switch. Doing this once for all models that potentially will be used on the Neural Compute Stick when the application starts running and saving the dimensions into a lookup table could cut the switch time almost in half. Further speedup of this switch could be achieved by conducting it asynchronously, as while the model is loaded onto the Neural Compute Stick the next frame can already be capture instead of waiting for the switching process to finish. All in all, I found that although at the current state this prototype would not be applicable to real time applications yet, given the potential for improvement it could get there. Yet if no hard conditions are imposed for it to perform in real time as is the case for many applications, it is deployable already.

PRACE SoHPCProject Title

Dynamic Deep Learning Inference on the Edge

PRACE SoHPCSite

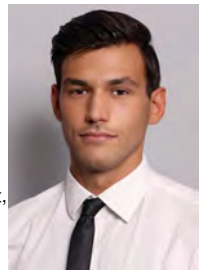
Irish Centre for High-End Computing, Ireland

PRACE SoHPCAuthors

Igor Kunjavskij, University of Stuttgart, Germany

PRACE SoHPCMentor

Venkatesh Kannan, ICHEC, Ireland


Igor Kunjavskij

PRACE SoHPCContact

Kunjavskij, Igor, University of Stuttgart
E-mail: Kunjavskij@gmx.de

PRACE SoHPCSoftware applied

Intel OpenVino

PRACE SoHPCMore Information

https://software.intel.com/en-us/openvino-toolkit

PRACE SoHPCAcknowledgement

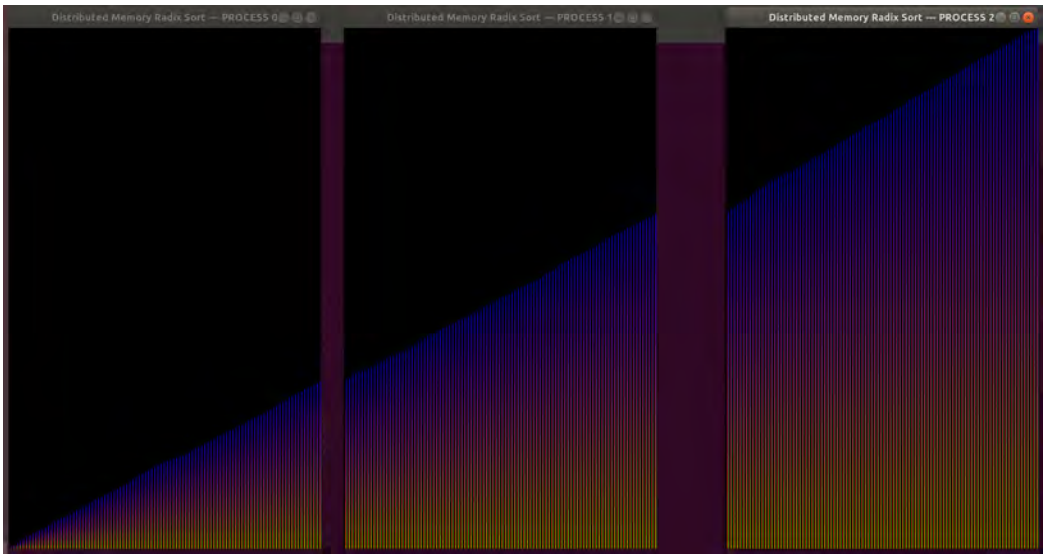
I would like to thank my supervisors for their amazing support throughout this whole project and in general the staff at ICHEC for welcoming me and making this stay such a great experience!

PRACE SoHPCProject ID

1914

Fastest sorting algorithm

Jordy Ajanohoun



With nowadays **large-scale data** and **large-scale problems**, **parallel computing** is a **powerful** ally and **important** tool. **Parallel sorting** is one of the important component in parallel computing. The parallel Radix sort allows us to **reduce the sorting time** and to **sort more data**, amounts that can't be sorted serially. Indeed, it is possible when we want to sort a huge amount of data that it can't fit on one single computer because **computers are memory bounded**. Or, it may take too much time using only one computer, a time we can't afford. Thus, for memory and time motivations, we can use **distributed systems** and have our data distributed across several computers and sort them in that way. But how?

Introduction

To make our applications and programs run faster, it is important to know where in our programs computers spend most of the execution time. Then we have to understand why and finally, we figure out a solution to improve that. **This is how HPC works**.
Donald E. Knuth wrote in his book **Sorting and Searching**: "*Computer manufacturers of the 1960's estimated that more than 25 percent of the running time of their computers was spent on sorting, when all their customers were taken into account. In fact, there were many installations in which the task of sorting was responsible for more than half of the computing time*".
In 2011, **John D. Cook, PhD** added: "*Computing has changed since the 1960's, but not so much that sorting has gone from being extraordinarily important to unimportant.*"

Indeed, it is become rare to work with data without having to sort them in any way. On top of that, we are now in the era of Big Data which means we collect and deal with more and more data from daily life. More and more data to sort. Plus, sorting algorithms are useful to plenty more complex algorithms like searching algorithms. Also, on websites or software, we always sort products or data by either date or price or weight or whatever. It is a super frequent operation. The question is how can we improve that? What kind of improvement can be done regarding sorting algorithms to go always faster? This is where the Radix sort and my project come into play.

It has been proved that for a **comparison based sort**, (where nothing is assumed about the data to sort except that they

can be compared two by two), the complexity lower bound is $O(Nlog(N))$. This means that you **can't** write a sorting algorithm which both compares the data to sort them and has a complexity better than $O(Nlog(N))$.
The **Radix sort** is a **non-comparison based sorting algorithm** that can runs in $O(N)$. Unfortunately, it is not so often used whereas well implemented, it is **the fastest sorting algorithm** for long lists. When sorting short lists, almost any algorithm is sufficient, but as soon as there is enough data to sort, we should choose carefully which one to use. The potential time gain is not negligible. It is true that "enough" is quite vague, but roughly, a length of 10000 elements is already enough to feel a difference between an appropriate and an inappropriate sorting algorithm.
Currently, **Quicksort** (a comparison based sort) is probably the most popular sorting algorithm. It is known to be fast enough in most of the cases, although its complexity is $O(Nlog(N))$. The Radix Sort has a complexity of $O(N * d)$. Thus, Radix sort is faster as soon as the number of keys digits (d parameter) is smaller than $log(N)$. This means **the more our list is huge, the more we should use the Radix sort**. To be more precise, from a certain length, the Radix sort will be more efficient than any comparison based sorting.

My project was to implement a reusable **C++ library** with a clean interface that uses Radix Sort to sort a distributed array using **MPI**.

Serial Radix sort

Radix sort takes in a list of N elements with their **sorting key** in a **base** b (the radix) and such that each **key** has d **digits**. For example, three digits are needed to represent decimal 255 in **base 10**. The same number needs two digits to be represented in **base 16** (FF) and eight in **base 2** (1111 1111). This is the **algorithm**:

Data: A (array to sort), N (the array size), b (the keys radix), d (the number of digits of the keys)
Result: A sorted according to the data keys
begin
  for *each* *key digit* ***i*** *where* ***i*** *varies from the Least Significant Digit* ***(LSD)*** *to the Most Significant Digit* ***(MSD)*** **do**
    sort A according to the i'th keys digit using **Counting sort** or **Bucket sort** ;
  end
end

Algorithm 1: Serial Radix sort

We first sort the elements based on the last keys digit (the Least Significant Digit) using Counting Sort or Bucket Sort. Then the result is again sorted by the second digit (from right to left), continue this process for all keys digits until we reach the Most Significant Digit, the last one on the left.

What if we want to sort a list of numbers having different number of digits in **base 10** like $(10, 149, 2, 46, 135, 25)$? **In practice**, to avoid this problem, we often use **base 256** to take advantage of the **bitwise representation** of the numbers in computers. Indeed, **a digit in base 256 corresponds to a byte**. Integers and reals are stored on a **fixed and known** number of bytes and we can access each of them. Therefore, the parameter d is fixed and known. For instance, we can write a Radix sort function which sorts int16_t (integers stored on two bytes) and we know in advance (while writing the sort function) that all the numbers will be composed of two base 256 digits. Plus, with templates-enable programming languages like C++, it is straightforward to make it works with all other integer sizes (int8_t, int32_t and int64_t) without duplicating the function for them. **From now, we assume that we use the base 256**.

Why use Counting or Bucket sort?

Because we take advantage of knowing all the values a byte can take and the range is small. The value of one byte is **between 0 and 255**. And this helps us because in such conditions, Counting Sort and Bucket Sort are simple and fast. They can run in $O(N)$ when the length of the list is greater than the maximum of the list (in absolute value) and especially when this maximum is known in advance. When it is not known in advance, it is more tricky to make the Bucket sort runs in $O(N)$ than the Counting sort. They can run in $O(N)$ because **they are not comparison-based sorting algorithms**, it is why the Radix sort too. In the Radix sort, we always sort according to one byte so the maximum we can have is 255. If the length of the list is greater than 255, which is a very small length for an array, Counting and Bucket sorts in Radix sort can easily be written having $O(N)$ complexity.

Why not use only either counting or bucket sort to sort the data all of a sudden?

Because **we will no longer have our assumption about the maximum** as we are no longer sorting according to one byte. The maximum of the list can't be known in advance and **we don't have an upper bound**. In such conditions, the complexity of Counting sort is $O(N + k)$ and Bucket sort can be worse depends on implementations. With k the maximum of the list (in absolute value). In contrast, with the Radix sort, we have $O(N * d)$ which is equivalent to $O(N)$ because d can't be greater than 8 (biggest numbers are usually stored on 8 bytes in computers). In other words, since the Radix sort iterates through the bytes and always sorts according to one byte, it is **insensitive** to the parameter k because we only care about the maximum value of one byte. Unlike both the Counting and Bucket sorts whose execution times are **highly sensitive** to the value of k, a parameter we rarely know in advance. It is dangerous because the maximum value can be a big number. For instance, if we have only 1000 numbers to sort and the maximum is 1000000, with Counting and Bucket sorts, the sorting time is more or less the same as sorting a list of 1000000 with them. Whereas it is not the case with the Radix sort, sort a list of 1000 elements using it will always take much less time than sorting 1000000 elements.

Problem the parallel Radix Sort can solve

The following is the problem more precisely define.

What we have

- P **processors** able to **communicate** between themselves, store and process our data. Each processor has a **unique rank** between 0 and $P - 1$.

- N data, **distributed equally** across our processors. This means that each processor has the same amount of data.

- The data are too huge to be stored or processed by only one processor (for memory or time reasons).

What we want

- **Sort the data across the processors**, in a distributed way. After the sort, the processor 0 should have the lower part of the sorted data, the processor 1 the next and so on.

- We want to be **as fast as possible** and **consume the lowest memory possible** on each processor.

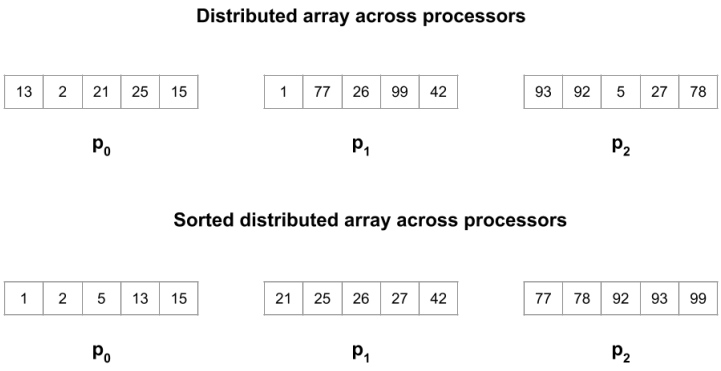


Figure 1: Unsorted versus sorted distributed array across three processors. Our goal with the parallel Radix sort is to get the sorted one from the unsorted.

Parallel Radix sort

The idea of the parallel Radix Sort is, **for each keys digit**, to first sort locally the data on each processor according to the digit. All the processors do that concurrently. Then, to compute **which processors have to send which portions of their local data to which other processors in order to have the distributed list sorted across processors and according to the digit**. Below is the algorithm.

Data: $rank$ (rank of the processor), L (portion of the distributed array held by the processor), N (the distributed array size), P (number of processors), b (the keys radix), d (the number of digits of the keys)
Result: the distributed array sorted across the processors with the same amount of data on each processor

begin
  for *each* *key digit* ***i*** *where* ***i*** *varies from the* ***LSD*** *to the* ***MSD*** **do**
    sort L according to the i'th keys digit using **Counting sort** or **Bucket sort**;

    share information with other processors to figure out which local data to send where and what to receive from which processors in order to have the distributed array sorted across processors according to the i'th keys digit;

    proceed to these exchanges of data between processors;
  end
end

Algorithm 2: Parallel Radix sort

Each processor runs the algorithm with its rank and its portion of the distributed data. Let's go through an **example** to understand better. We will run the example in **base 10** for simplicity but **don't forget** that we can use any number base we want and in practice, base 256 is used as explained.

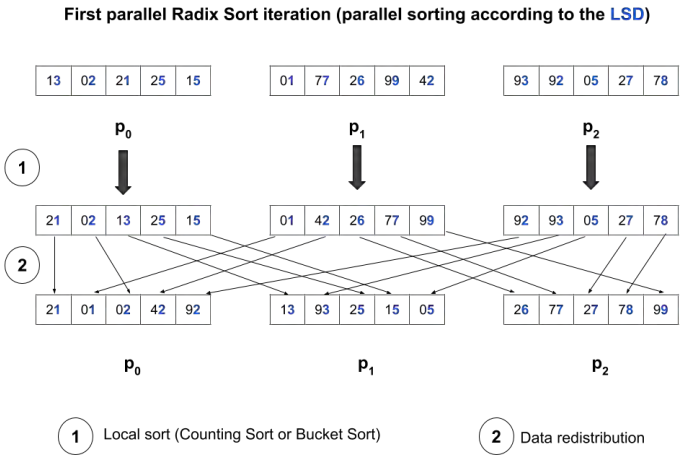


Figure 2: First iteration of the parallel Radix Sort on the distributed array across three processors. The numbers are sorted according to their base 10 LSD (Least Significant Digit). One iteration according to their base 10 MSD is remaining to complete the algorithm and get the desired final distributed sorted array.

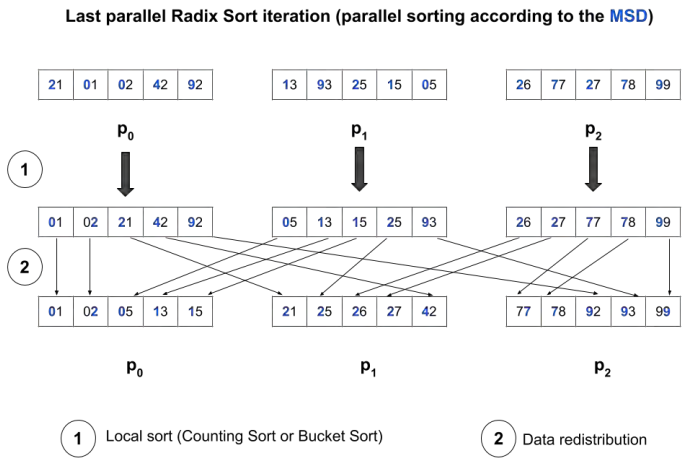


Figure 3: Last iteration of the parallel Radix Sort on the distributed array across three processors. The numbers are sorted according to their base 10 MSD (Most Significant Digit). At the end, the algorithm is completed and the distributed array is sorted.

My implementation and performance results

I have used **MPI** for the communications between processors. As hardware, I have used the **ICHEC Kay cluster** to run all the benchmarks. The framework used to get the execution times is **google benchmark**. The numbers to sort are generated with **std::mt19937**, a Mersenne Twister **pseudo-random generator** of 32-bit numbers with a state size of 19937 bits. For each execution time measure, I use **ten different seeds** (1, 2, 13, 38993, 83030, 90, 25, 28, 10 and 73) to generate ten different arrays (of a same length) to sort. Then, the execution times **mean** is registered as the execution time for the length. These ten seeds have been chosen randomly. I proceed like that because the execution time also depends on the data, so it allows us to have a more accurate estimation.

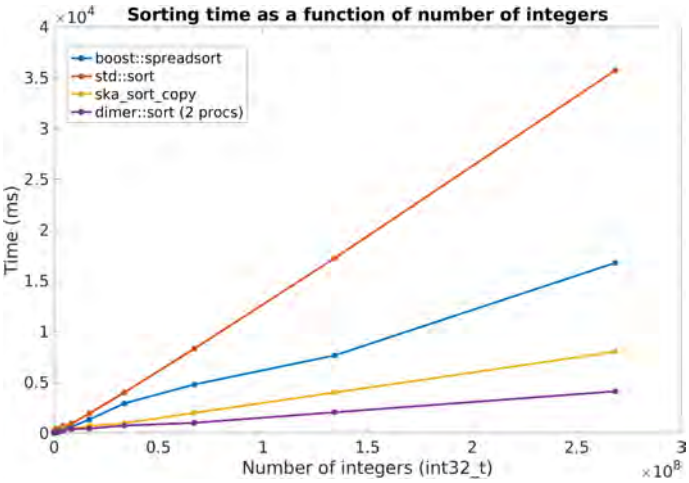


Figure 4: Sorting time as a function of the number of items to sort. The items here are 4 bytes integers. dimer::sort curve is the execution times of my parallel Radix Sort implementation ran with 2 processors.

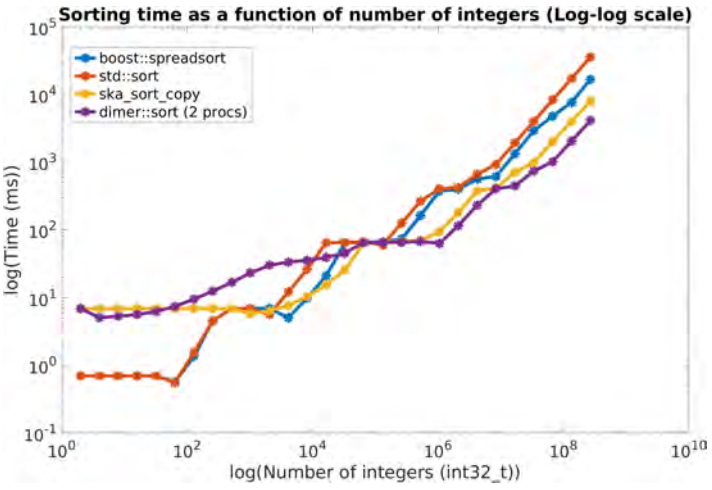


Figure 5: Same as figure 4 but in log-log scale.

We can notice that my implementation seems good for big sizes. It seems to be faster than boost::spreadsort because when the array is not distributed, I am using ska_sort_copy instead of boost::spreadsort. ska_sort_copy is a **great** and **very fast** implementation of the serial Radix sort. Actually, **it is the fastest I have found**.

With the log-log scale we can see better what is happening. We are able to tell that using dimer::sort we go faster than anything else when the array is enough big and this is what we expect when we use HPC tools. For small array sizes, we are not really expecting an improvement because they can be easily well managed serially and enough quickly. Plus, we add extra steps to treat the problem in a parallel way, therefore, when the problem size is small, it is faster serially because we don't have these extra steps. It becomes faster in parallel when these extra steps are a small workload compare to the time needed to sort serially.

Future work

I have presented here the **LSD** Radix Sort. But, there is a variant called **MSD** Radix Sort which can be parallelized too. The difference is:

- With the **LSD**, we sort the elements based on the keys LSD first, and then continue to the left until we reach the MSD
- With the **MSD**, we sort the elements based on the keys MSD first, and then continue to the right until we reach the LSD

For more information, feel free to read my blog posts on PRACE Summer of HPC website. My project is more detailed there and you can find everything you need to reproduce the work or understand deeper.

Acknowledgement

I would like to express my gratitude to the whole ICHEC staff, with special thanks to Dr. Paddy Ó Conbhuí and Dr. Simon Wong for supervising the project. Special thanks go to PRACE for giving me a place on this Summer of HPC 2019. Many thanks to Dr. Leon Kos and Dr. Massimiliano Guarrasi for all their work with the training week and the program in general.

References

[1] A. Sohn and Y. Kodama, Load balanced parallel radix sort,in "Proc. 12th ACM International Conference on Supercomputing, Melbourne, Australia, July 14–17, 1998".

PRACE SoHPC Project Title

Distributed Memory Radix Sort

PRACE SoHPC Site

ICHEC (Irish Centre for High-End Computing), Dublin, Ireland

PRACE SoHPC Authors

Jordy Ajanohoun
Sorbonne University, Paris, France

PRACE SoHPC Mentor

Paddy Ó Conbhuí
ICHEC, Dublin, Ireland


Jordy Ajanohoun

PRACE SoHPC Contact

Jordy, Ajanohoun, Sorbonne University
Phone: +33 6 95 39 87 90
E-mail: jordy.ajanohoun@hotmail.fr

PRACE SoHPC Software applied

C++, MPI, Catch2 (C++ test framework), google benchmark, Slurm

PRACE SoHPC More Information

https://summerofhpc.prace-ri.eu/author/jordya/

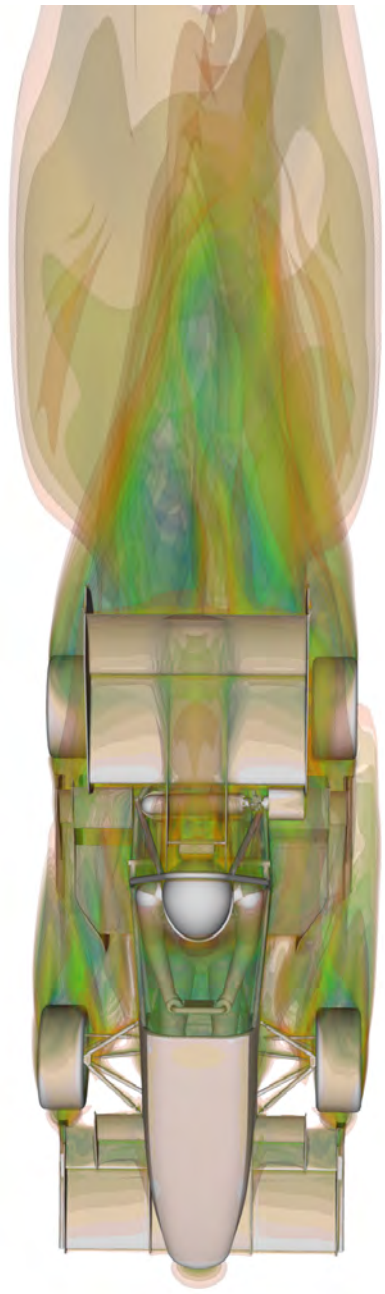
PRACE SoHPC Project ID

1915

Study on how HPC systems can enhance and interact with CFD at all stages: pre-processing, resolution and visualization

CFD: *Colorful Fluid Dynamics?* No with HPC!

David Izquierdo Susín



It is typical to obtain some nice pictures of the flow around an object after performing a CFD (Computational Fluid Dynamics) simulation. However, how much of truth is there behind those colorful images? The project presented here analyses **how HPC systems can be used to make CFD a more trustworthy tool**, and shows how some parameters can be optimized for a specific application - a formula car.

ABOUT FLUID DYNAMICS

The **Navier-Stokes** equations are the set of equations that define how a fluid moves, and are based on principles such as the **conservation of mass and Newton's Second Law**. When one is set to solve an aerodynamics problem, solving these equations will always be, somehow or other, unavoidable. The *only* problem is that these equations are actually **impossible to solve** analytically in the vast majority of practical cases, and the accuracy of the results heavily depends on the numerical method used.

Aerodynamics is about computing the forces that the air causes on a given object -in this case, a **Formula Student car**-. To get the forces, one must first compute the pressure at many points over the surface of the object, and the pressure cannot be obtained without computing the velocity in the whole domain; the problem is coupled.

When solving the motion of a fluid around an object, the objective is, therefore, to know the **velocity and the pressure of the fluid at many different points**. In order to obtain a result similar to that shown below in (5), where the **scale from blue to red shows the values of the velocity** on the symmetry plane of the car, the variables must be solved at a very large amount of points, in order to then be able to *paint* each of these points with a different colour, corresponding to the value of the velocity at that point.

For a case like the Formula Student car, around 100 000 000 of these points are required to get an accurate enough solution. If it is estimated that the computer needs more than 100 operations to solve a value single point/cell (the equations are complex), and it is assumed that we want the value of the velocity at 1 000 time instants -to observe how the flow evolves as time passes by-, then the total number of computer operations may be in the order of **10 000 000 000 000**, or what is the same, ten of the former UK billions.

All this constitutes just a rough order of magnitude approximation to give an idea of the size of the problem that is being faced and its motivation.

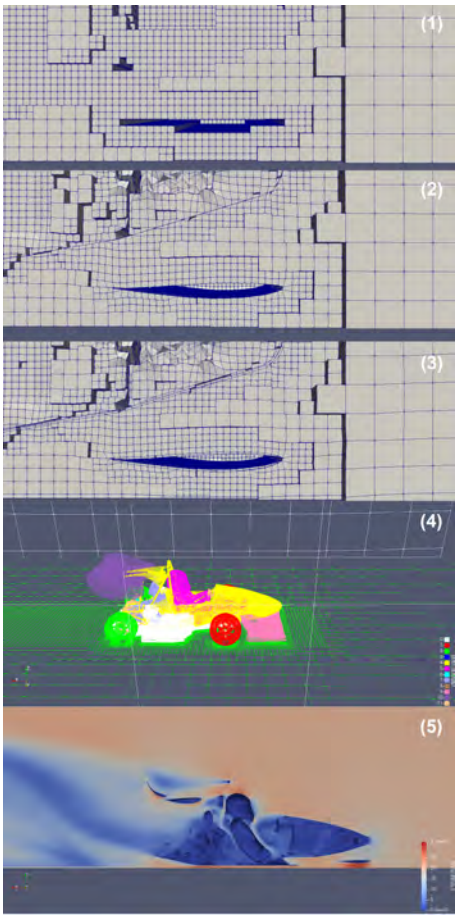


Figure 1: Visual state of the geometry at some of the steps in the CFD workflow.

HPC IN OSTRAVA

Here is where **HPC (High Performance Computing) systems come in handy**. HPC systems are essentially computers that consist of many smaller computers that are able to work together to solve a specific task. When such a large number of operations is required to solve just once the problem of interest, using a personal computer is not an option: it would not only be **too slow** -weeks or even months would be needed to get the first results of a complex problem-, but possibly it would be incapable of solving the problem, due to the large amount of data that must be handled, which in turn implies a **very large requirement in terms of memory**.

The IT4Innovations National Supercomputing Centre in Ostrava offers two of these very powerful systems that can be used to better solve these kind of problems. In particular, Anselm machine relies on more than 3 300 cores, while Salomon has up to 24 192 cores. To put it into context, a typical laptop can typically be based on 4 processors and 8 logical units, which is more **than 3 000-fold less than Salomon**.

WHAT IS THIS TRULY ABOUT?

The engineering problem to be dealt with in this document is the **aerodynamic analysis of the TU Ostrava Formula Student car**. The main objective is building from scratch a **robust simulation setup** with which obtaining results for the aerodynamic forces on the Formula Student car in an **automated** manner. Apart from that, it is the aim of this project to better **understand the capabilities of OpenFOAM** when dealing with external aerodynamics cases.

METHODS (HANDS-ON)

The approach used to solve the tasks mentioned in the previous section consists of using the **Linux** operating system to install the corresponding software and edit the files in a more efficient manner than what could be achieved in a Windows operating system, by means of *vim* and other editors like *sed*.

The methodology followed in this project is defined in several steps that can be easily distinguished one from another:

- Pre-processing of the geometry
- Meshing setup
- Simulation setup
- Post-processing of the results

The first step consists of getting **the initial car geometry** designed by the TU Ostrava Formula Student team and **transforming** it into a surface mesh of points that can be read by the CFD software. Then, a proper **mesh has to be generated**, in this case by means of the *snappyHexMesh* utility. There are **more than 50 parameters** to be adjusted, and the suitability of the mesh heavily depends on the choices made on these parameters. Above at the left, in the images (1) to (3) of Fig. 1, the different steps performed when creating a mesh can be observed: in (1) the mesh after the first step (*castellation*) is shown. It can be seen that the mesh is not adapted to the surface geometry of the car, since it is just made out of cubes. This is solved in the next step (2), in which the mesh is snapped to provide it with the actual shape of the geometry. Finally, in step (3) *layers* are added near the surface of the body. This type of cells are beneficial when trying to solve the behaviour of the flow in the boundary layer.

In the following step, it is necessary to define the **physical and numerical parameters** on which the simulation will be based. The choices made at this point will determine the degree of convergence of the simulations, together with the accuracy of the results. Again, the number of parameters is very large and it is of significant difficulty to find the ideal configuration. The approach followed to obtain an stable setup without the need of adjusting one by one all the parameters was to pick as a base a reference case provided by the software and based on a motorbike simulation. In this way, even if the parameters had to be adjusted (since the geometry proposed for the project was far more complex, and the requirements in terms of accuracy were also more restricting), the process was simplified to some extent.

Lastly, the huge bunch of numbers obtained as an output from the simulations must be **post-processed** in order to gain some degree of comprehension from the simulations performed. This was done as an initial checking step in ParaView, an example of which can be seen in (5) above, but later the process was migrated to EnSight, where further and more involved post-processing was conducted, including generation of iso-surfaces and videos.

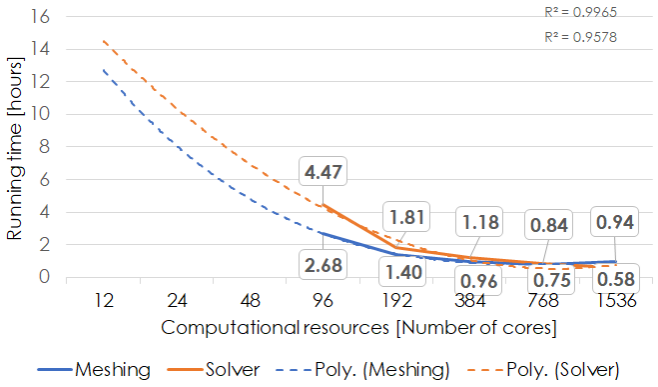


Figure 2: Scalability Curves

Given the complexity of the setup of the simulations in terms of number of

parameters to be defined, it is not possible to show or discuss here the full set of parameters. But, in order to ensure **that the work is reproducible**, some extra detail is given in this blog post and in the final presentation.

SCALABILITY AND MESH CONVERGENCE

The first outcome of this project consists of the **scalability study**. This study is aimed at determine how OpenFOAM works depending on the number of cores used to run the cases. Results are shown in Fig. 2, where it can be observed that the solving algorithm (orange curve) scales better than the meshing algorithm (blue curve). In the case of the former, between 768 and 1536 the reduction in total computing time is still of a 31.0 %, which is far from the ideal 50 % but it is still significantly large given the huge amount of cores.

Two approaches were considered to run the simulations: a single-job approach, in which a single job would be launch for each simulation, and a two-jobs approach, that separates the meshing and solution algorithms execution into two different scripts and two different jobs.

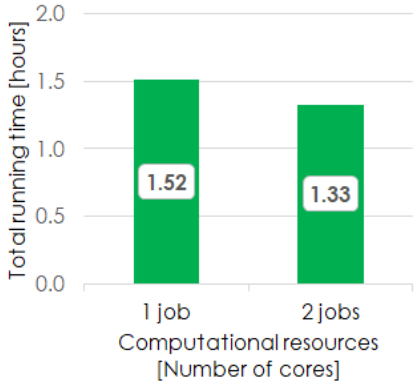


Figure 3: Single-job Versus Two-jobs Approach

The results in Fig. 3 show that the **two-jobs approach**, finally adopted in the project, is a **12 % faster**, which is a significant reduction that does have an impact, since it allows a more optimal allocation of resources.

All these computations were performed once the **mesh convergence had been proved**. This is very important, since it ensures that the results of the aerodynamic study are approximately independent of the size of the mesh elements. It was achieved via a mesh convergence study for which first several meshes are defined in Fig. 4.

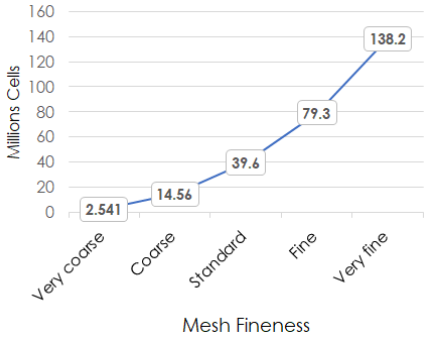


Figure 4: Reference Meshes

For the reference meshes with the number of cells shown in Fig. 4, the mesh quality characteristics depicted in Fig. 5. have been found.

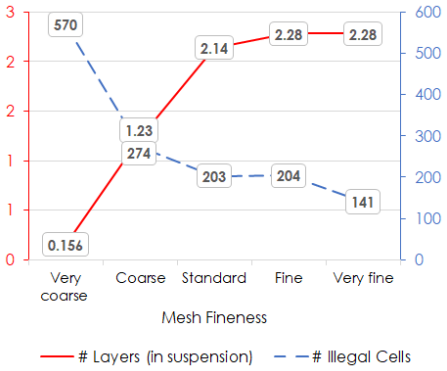


Figure 5: Monitoring of Layers Creation and Quality Check on Meshes

In terms of the number of layers (referring again to the step (3) in Fig. 1 for the layer definition), the target was set to three. One of the geometries in which the **creation of layers is more complex, due to its geometry, is the suspension** of the car. Therefore, the number of layers is monitored in this part and it can be checked that, for the finer meshes, the user requirement is almost met. In terms of illegal cells, that are the cells that **do not pass the user-selected quality check**, the number was also significantly reduced with larger meshes -although note that quality standards were set very strict to perform this evaluation; not all cells that did not pass are necessarily badly-shaped-.

Next figure, Fig. 6, shows the results of the mesh convergence study. **Several coefficients of forces were monitored** for the different mesh configurations with the already described characteristics. In particular, these coefficients are the total coefficient of lift (in absolute value, it is actually negative), the total coefficient of drag of and the rear coefficient of lift of the car, and the coefficients of lift and drag of the front wing. By observance of the fact that the **value of the coefficients tends to become constant** as the mesh gets finer, it may be concluded that the **results are mesh independent** for the roughly 140-millions mesh used for the scalabitily report.

However, Fig. 6 also shows that, even if the variations tail off, these do not totally disappear, and in fact in some of the coefficients (such as the total CL) these variations seem to slightly increase with the step from *fine* to *very fine* mesh with respect to the previous steps.

All this may indicate that convergence has not been achieved up to a very high degree, and this is expected to have something to do with the boundary layer resolution and the y^* value -non-dimensional height of the centroid of the cells touching the body surface; i.e. size of the cells close to the surface-, which should lie everywhere below six, but that is not true for the whole mesh.

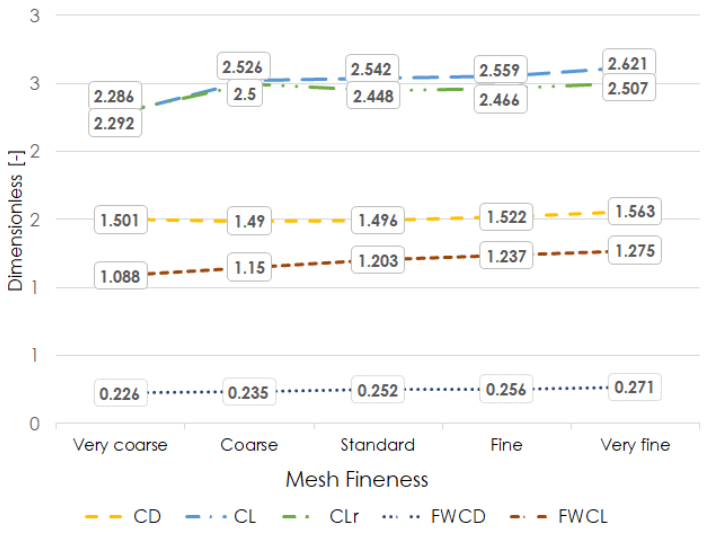


Figure 6: Mesh Convergence Study

In fact, y^* goes up to 17-20 in some regions, making thus necessary the use

of wall functions to cover the delicate buffer layer and losing some of the advantages of the k-omega SST turbulence model[1] used in these simulations.

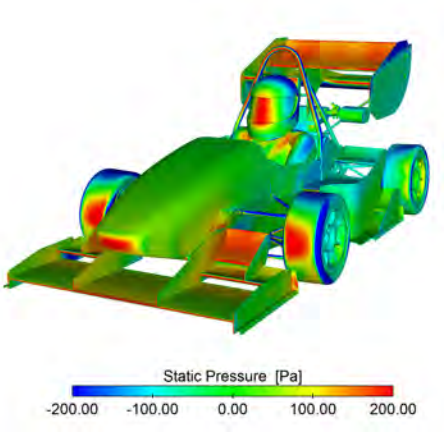


Figure 7: Static Pressure over the Surface of the Car

The quality of some of the cells may also be a reason for this imperfection in the mesh convergence results.

In spite of that, the results are reasonable in overall terms and **meet the accuracy requirements that had been proposed**. This allows to proceed to the next part, in which the actual post-processing results are going to be discussed by means of images of the car.

POST-PROCESSING

The final study entails analysing the flow around the car, which is done by looking at the results in terms of velocity and pressure fields.

Fig. 7 illustrates the pressure field over the surface of the car. It is easy to observe that the more drag-generating parts -largest static pressure, in red- are those exposed in a more perpendicular and direct way to the flow, such as the tip of the nosecone, the driver, the front tyres and the steepest zones of the wings. The zones of smallest static pressure (negative with respect to the ambient pressure, i.e. suction) lie below the car, specifically below the front wing and the undertray, thanks to the suction capabilities of those devices, although these cannot be seen in this image. What can be seen is the reduction in pressure in the zones of high curvature, where corner flows emerge. Examples of this are the lateral of the front wheels and the lateral of the helmet of the driver. The triangular blue zone in the rear wing end-plates is also relevant, since it shows how the wing-tip vortex that arises at the edges of every wing develops, growing towards the rear end of the wing.

Finally, in Fig. 8 the wake of the car can be observed from the bottom. It is constructed by plotting **velocity isosurfaces** for a range of values of the velocity going from $2\ m/s$ up to $15\ m/s$ (with the free stream always $20\ m/s$, therefore all the values plotted are smaller).

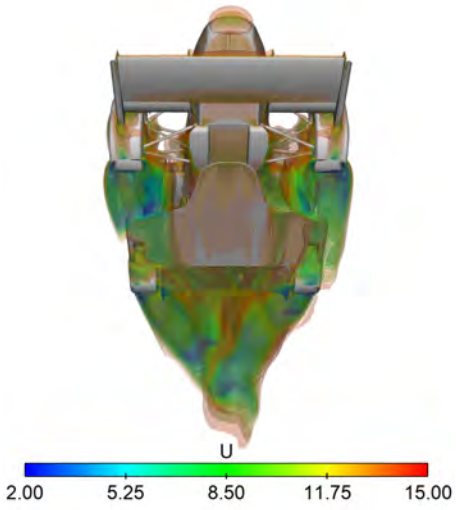


Figure 8: Wake of the Car - Bottom View

A first remark to be made is that the **flow is not symmetry, since the geometry is not symmetric**. Even if the car may look as if it had symmetry at first sight, it turns out that the intake of the engine and the cooling system (radiator and fan) and highly non-symmetrical, which leads to a wake slightly deflected towards the left, from a driver's perspective. Apart from that, the vortical structures created around each of the four tyres are of extreme importance for the whole aerodynamic vehicle concept due to its influence in all the other aerodynamic devices of the car, and therefore, this kind of representations are essential to understand them and manage them properly.

DISCUSSION

Overall, the objectives of the project were met and key conclusions were drawn: (1) OpenFOAM solver *simpleFoam* scales well -while the meshing utility is the bottleneck-, (2) the two-jobs approach is better than the single-job one and (3) using HPC systems helps to reach more meaningful and trustworthy results with CFD for a same amount of time available. The implications of these results are that a better and more extensive knowldege about the use of OpenFOAM in HPC systems will be available for the development of similar projects.

References

[1] F. R. Menter, M. Kuntz and R. Langtry (2003). Ten Years of Industrial Experience with the SST Turbulence Model.

PRACE SoHPCProject Title

Computational Fluid Dynamics Simulations of Formula Student Car Using HPC

PRACE SoHPCSite

IT4Innovations - National Supercomputing Centre. Ostrava, Czech Republic.

David Izquierdo Susín

PRACE SoHPCAuthors

David Izquierdo Susín, UC3M, Madrid

PRACE SoHPCMentor

Tomáš Brzobohatý, VŠB, Czech Republic

PRACE SoHPCContact

David Izquierdo, UC3M
Phone: +34 658 43 09 55
E-mail: davidizquierdosusin@gmail.com

PRACE SoHPCSoftware applied

OpenFOAM, ParaView, EnSight

PRACE SoHPCMore Information

www.openfoam.org

PRACE SoHPCAcknowledgement

I would like to thank all the people from PRACE for organizing the programme, as well as the Ostrava coordinators and my mentor, Tomáš Brzobohatý. And above all, I wanted to thank my family, and all the people that helped me to get here.

PRACE SoHPCProject ID

1916

Real time emotion recognition based on action units using deep neural networks (DDNs) and clustering algorithms. From High Performance Computing (HPC) to the edge.

Emotion Recognition using DNNs

Pablo Lluch Romero

Visual context, particularly facial expressions, play a key role in understanding the intention of a message. However, visually impaired individuals lack this context. An application able to detect 7 basic emotions in real time with an accuracy of 74% is introduced to compensate for the lack of visual context in conversations.

In order to make a machine understand facial expressions we need a way to encode an image, an array of pixels, such that its emotion is easily classifiable. Many different approaches can be used to make this encoding but, I will be using a facial action coding system that classifies sets of muscle movements in bundles called action units. Any facial expression can be encoded using this framework. However, I used 17 different action units, as shown in figure 1, which are enough to encode the emotions I trained the application to predict: happiness, sadness, surprise, anger, disgust, fear and neutral.

I used a deep neural network to predict the action units from faces. A neural network is a computer algorithm that imitates biological neurons with many interconnected layers that is able to find underlying patterns in data. A nearest neighbour approach is used from the output of the trained network to classify new images as a part of the final application.

Figure 1: Action units used[1]

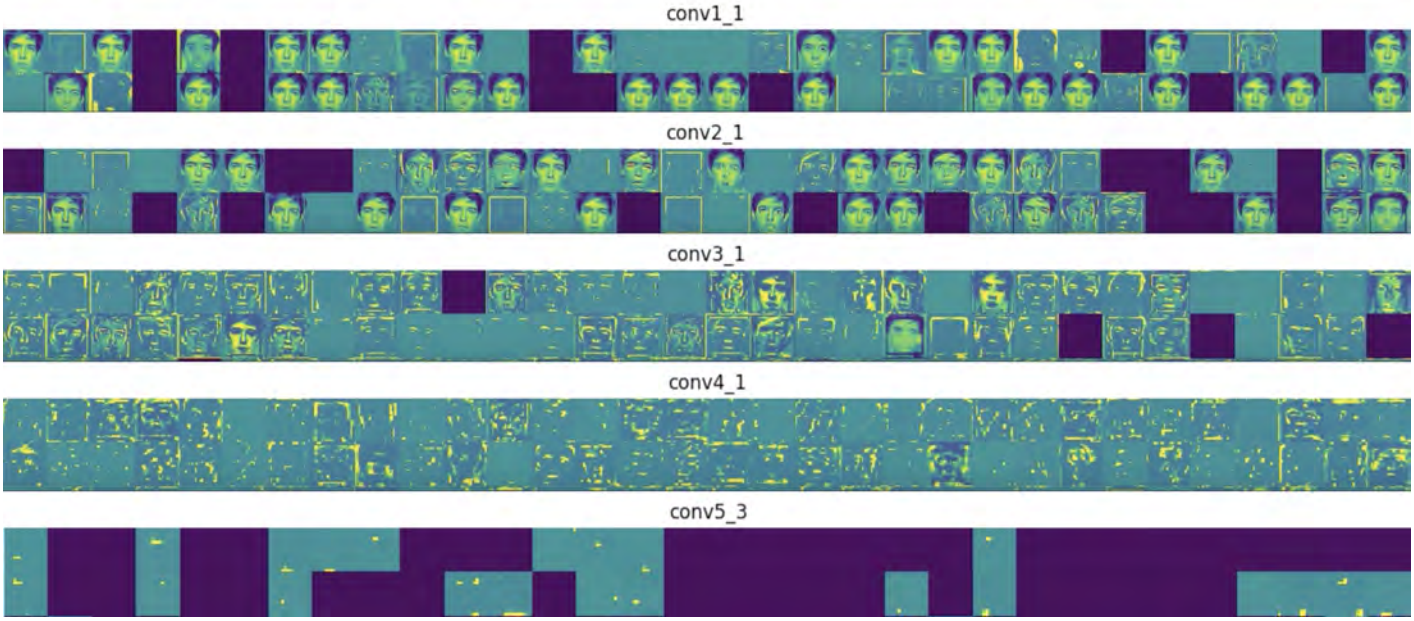


Figure 2: Sample layers from superficial [top] down to deeper ones [bottom] from fine-tuned VGGFace

From HPC...

Data gathering and preprocessing

The data I needed to train the neural network came from four existing datasets that consisted of video footage of spontaneous emotions with the active action units annotated frame by frame. These were The Cohn-Kanade AU-Coded Facial Expression dataset[2,3], MMI facial expression database collected by Valstar and Panticand[4,5], the Denver Intensity of Spontaneous Facial Action Database[6,7], and BP4D-Spontaneous[8,9]. The choice of datasets was not trivial, as they had to be consistent with the way the actions units were annotated.

Once I combined these four datasets into one with a total of 300K images, I went through an array of different processes to improve its quality. The first issue that I faced was class imbalance. The most popular action unit in the combined dataset (upper lip riser) was present in 80,000 images, while the least popular (lip presor) was in only 3,000. Using such unbalanced data to train a machine learning model is a very bad practice as these models learn the statistical footprint of the classes. So, very popular classes would yield a lot of false positives and less popular ones a lot of false negatives. Secondly, the contiguous frames were removed from the dataset to prevent the model from overfitting –memorising the individual patterns of a dataset without gaining the ability to generalize on unseen data. Finally, in order to increase the size of the dataset and add more variability to the classes I augmented the data by applying random rotations, shears, zooms, and brightness shifts to each image before feeding it into the network, as shown in Figure 3.

Figure 3: Data augmentations from a single image ©Jeffrey Cohn

Training

I built the Neural Network using Transfer Learning. This is a common practice that involves reusing existing networks that were devised to solve problems in a similar domain. I took the convolutional layers from a network called VGGFace, which was originally developed for face detection. A convolution can be understood as a filter that is applied to an image. So, convolutional layers are just a set of these filters acting both in parallel and in sequence. Superficial convolutional layers encode low level concepts like edges or corners and deeper ones encode meaning related to the class of the image. A sample of these layers are shown in figure 2. I also fine-tuned the weights of these convolutional layers to match the problem of action unit detection better. The output of the last convolutional layer (conv5_3 as shown in Figure 2) could be thought of as the high-level features that define a face. In order to classify these features I added an array of dense and dropout layers to the network. Dense layers are fully connected layers of neurons that fire an output only if their inputs are high enough, the same way neurons behave in our brain. Dropout layers disable a random fraction of the connections between neurons during the training process.

During the training of the neural network, I explored a number of hyperparameters and settings such as the learning rate, the batch size, the number and size of dense and dropout layers, and different preprocessing of the data. I found that using dropout layers before and after every dense layer reduced overfitting considerably. However, the model achieved the best validation accuracy when the first dropout layer kept a larger portion of units than the others. The learning rate –a parameter that specifies how fast a model reacts to new information– was also key. Too high or too low of a learning rate wouldn't let the model arrive to a decent solution in a reasonable time. I also found out that the larger the size of the dense layers, the lower the learning rate had to be for the model to converge. The

data preprocessing mentioned before also reduced overfitting, as it increased variability and removed similar images. The final setting produced a model able to identify action units with a recall of 64.5% and a precision of 73.6%. For the training process described above I used Keras and Tensorflow as the environment, and I ran it on the NVIDIA DGX-2 cluster of the IT4Innovations supercomputer center. I only used a single slice (a single GPU) of the DGX-2 but that was already 30x faster than the CPU of my development system.

Figure 4: Topology of the final neural network that uses VGGFace's convolutional layers

... to the edge

Inferring the emotions in real-time

The action units predicted using the deep neural network were translated to the corresponding emotion using a nearest neighbour approach.

The peak emotion images in the Cohn-Kanade dataset were fed into the trained neural network and the output –the probability of the image containing each action unit– was stored in a small dataset of around 300 images. Kohn-Canade also included the corresponding emotions, so I stored this alongside the action unit prediction. Inferring the emotion that corresponds to a new image is just a matter of getting the vector of action unit probabilities from the network, finding the most similar probability vectors in this small dataset and looking at the most common emotion among them. This could be understood as looking at the neighbours of the action unit probability vector in a hyper-dimensional space where each dimension corresponds to one of the 17 action units. Another more straightforward approach would be checking the action units whose probabilities are larger than 0.5 and comparing them to a table of the common action units that are archetypal in each emotion. However, the approach I followed is better for two reasons: A) it allows for a wider variability in the emotions, as not every emotion fits within these prototypical definitions and B) it helps mitigate biases that the model might have. Even if the action unit prediction is not perfect, as the small dataset was created using this biased model, the imperfection will be accounted for when the new prediction is compared to the existing points in the hyperspace.

Application pipeline

Faces are cropped from a live camera feed using Python OpenCV's Haar Cascade Classifier face detection algorithm running at approximately 60ms per image. Images are downscaled before they're fed into the algorithm for speedup reasons and then the face position is inferred in the original image so no resolution is lost. Faces are fed asynchronously to the trained model that predicts the action units only once the model has finished processing the previous image. The action units predicted by the model are then compared to the small dataset of Kohn-Canade predictions to find the 10 closest neighbours using cosine similarity, and the most common emotion among them is chosen.

Both cosine similarity and euclidean distance were tried but cosine similarity yielded a better result. In an ideal world the neural network would only return values very close to 0 (when the action unit is not present) or values very close to 1 (when the action unit is present at any level of intensity), as a sigmoid function is used on the final layer of the DNN to polarise the probabilities.

However, this is not the reality for my model. The reason for why cosine similarity gave a better result is that all we care about for classifying an emotion is the ratios between the different action units, not the size of the action unit vector itself –as this would encode the intensity of the emotion. Using euclidean distance would result in different intensities considered as different emotions, which would make it harder to classify new predictions.

This approach needs a way to determine when no emotion is present (or the emotion is neutral). For this I set a threshold in the probabilities of the action units so that if every action unit is below this threshold then the emotion had to be neutral. The emotion displayed in the application is the most common emotion over the last five inferences (i.e. frames) to avoid flickering and foster stability. In order to produce the audio saying the emotion, I used python's gTTs text to speech module. The audio message is only fired once the emotion has settled to avoid constant interruptions.

Figure 5: 2D representation of the emotion hyperspace corresponding to the action unit predictions

The original idea of the project was to deploy the application to the Intel Movidius Neural Compute Stick 2 (NCS2). This was achieved using OpenVINO and Intel's own model optimizer. A working application was made but the NCS2 was giving incorrect inference results, probably due to the lack of

compatibility with some neural network operations. The older NCS 1 was used instead, yielding an inference time of 900ms. Another version of the application was deployed using the CPU of my development system where Keras and Tensorflow were used for the inference. This version was able to infer the emotions in 200ms.

but a sequence of them. An objective metric of the accuracy of the application is still a matter for further analysis. This just serves as a quick verification of how the application is able to infer the emotion on different and diverse subjects.

During the training process of the neural network it's important to have a

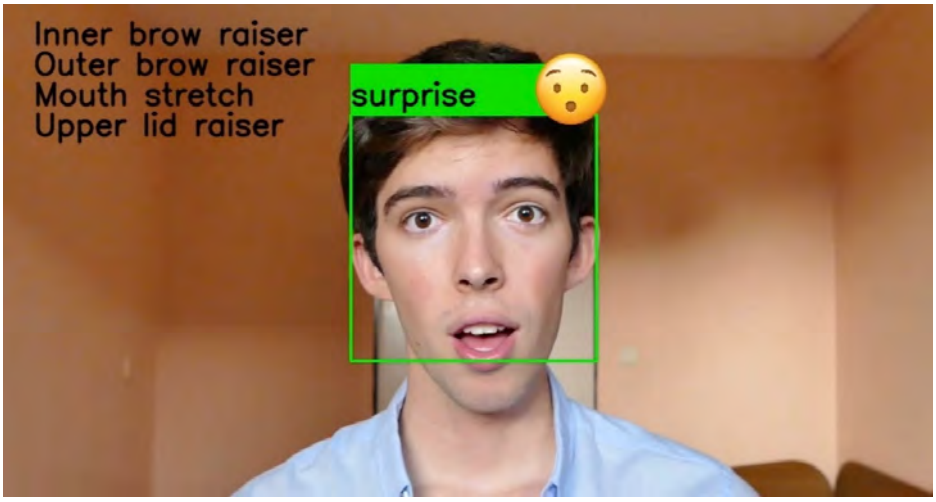


Figure 6: Screenshot of the working application with the action units showing at the left

Discussion & Conclusion

I believe a very important part of this work is its scalable nature. Making the application detect a new emotion wouldn't require any further training of the neural network as long as the action units present in the emotion are included in the 17 action units which the network is able to detect. All that would be needed is to add a few new data points corresponding to the predictions of the model on images containing that new emotion to the small action unit-emotion dataset that I used for the multi-dimensional comparison.

In order to assess the accuracy of the application, I compiled a dataset of images taken from Google corresponding to the 7 target emotions. This dataset contains faces with clearly identifiable unambiguous emotions in good lighting and resolution. It includes different ages, genders and ethnicities. The application predicted the correct emotion on this dataset with an accuracy of 74%. The app doesn't perform as well with images that are ambiguous, unclear or contain cues other than muscle movements (for example the application wouldn't categorise a person with tears in their eyes as sad). It's important to bear in mind that the live application delivers a slightly better accuracy than the individual predictions, as not only one picture is considered to determine the emotion robust metric of how good a model is. This is true because multiple combinations of hyperparameters are tried during training, so multiple versions of the same model are produced. Even though the neural network is being trained to detect action units in faces, the ultimate goal of these action units is to serve as a basis to infer the emotion on images. Thus, I will suggest a way to evaluate a model by examining how well these action units serve to predict emotions. If we go back to the action unit hyperspace that was discussed before, it can be seen that emotions form clusters in it. A good way of assessing how easily separable these emotions are is to evaluate how little the clusters overlap with each other. Thus, very separate and isolated clusters mean easily classifiable emotions and higher accuracy overall. That's why, a metric to evaluate this lack of overlap could be introduced to evaluate the possible models and determine which will yield the best accuracy in the final application. Another possible optimization would be to reduce the dimensionality of the small action unit-emotion dataset to reduce the time taken to find the nearest neighbours. This could be done using algorithms like PCA that, as shown in Figure 5, are able to reduce data dimensionality while still showing separable clusters, which as explained above, translate to easily classifiable emotions.

References

[1] Facial Action Coding System, accessed 30th/August/2019, imotions.com/blog/facial-action-coding-system/

[2] Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis. Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00), Grenoble, France, 46-53.

[3] Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., & Matthews, I. (2010). The Extended Cohn-Kanade Dataset (CK+): A complete expression dataset for action unit and emotion-specified expression. Proceedings of the Third International Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB 2010), San Francisco, USA, 94-101.

[4] M.F. Valstar, M. Pantic, "Induced Disgust, Happiness and Surprise: an Addition to the MMI Facial Expression Database", Proceedings of the International Language Resources and Evaluation Conference, Malta, May 2010

[5] M. Pantic, M.F. Valstar, R. Rademaker and L. Maat, "Webbased database for facial expression analysis", Proc. IEEE Int'l Conf. on Multimedia and Expo (ICME'05),Amsterdam, The Netherlands, July 2005

[6] Mavadati, S.M.; Mahoor, M.H.; Bartlett, K.; Trinh, P.; Cohn, J.F., "DISFA:A Spontaneous Facial Action Intensity Database," Affective Computing,IEEE Transactions on , vol.4, no.2, pp.151,160, April-June 2013 , doi:10.1109/T-AFFC.2013.4

[7] Mavadati, S.M.; Mahoor, M.H.; Bartlett, K.; Trinh, P., "Automatic detection of non-posed facial action units," Image Processing (ICIP), 2012 19th IEEE International Conference on , vol., no., pp.1817,1820, Sept. 30 2012-Oct. 3 2012 , doi: 10.1109/ICIP.2012.6467235

[8] Xing Zhang, Lijun Yin, Jeff Cohn, Shaun Canavan, Michael Reale, Andy Horowitz, Peng Liu, and Jeff Girard, "BP4D-Spontaneous: A high resolution spontaneous 3D dynamic facial expression database", Image and Vision Computing, 32 (2014), pp. 692-706 (special issue of the Best of FG13)

[9] Xing Zhang, Lijun Yin, Jeff Cohn, Shaun Canavan, Michael Reale, Andy Horowitz, and Peng Liu, "A high resolution spontaneous 3D dynamic facial expression database", The 10th IEEE International Conference on Automatic Face and Gesture Recognition (FG13), April, 2013.

PRACE SoHPCProject Title

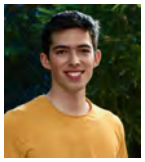
Emotion Recognition using DDNs

PRACE SoHPCSite

IT4Innovations, VŠB - Technical University of Ostrava, Czech Republic

PRACE SoHPCAuthors

Pablo Lluch Romero, The University of Edinburgh, United Kingdom



Pablo Lluch Romero

PRACE SoHPCMentors

Georg Zitzlsberger and Martin Golasowski, IT4Innovations, Czech Republic

PRACE SoHPCContact

Leon, Kos, University of Ljubljana
Phone: +12 324 4445 5556
E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied

Python, Tensorflow, keras

PRACE SoHPCAcknowledgement

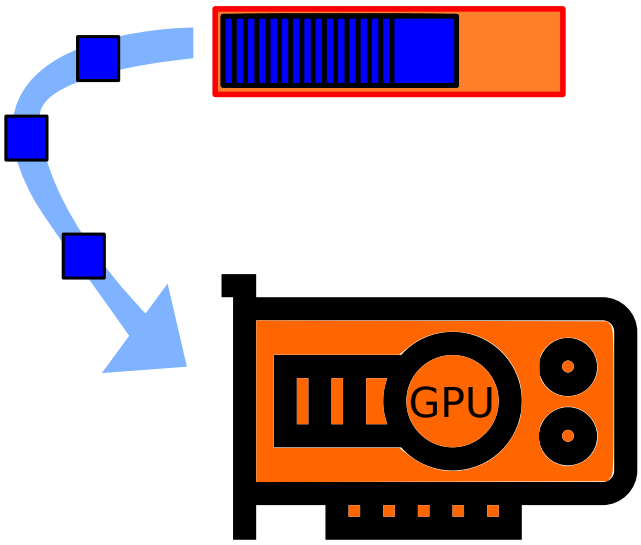
Thanks to IT4Innovations and my mentors Georg Zitzlsberger and Martin Golasowski for providing me with access to the supercomputer facilities and support throughout my project. Also thanks to PRACE for creating this opportunity and allowing us to get closer to the incredibly interesting and power field of HPC. I would also like to thank the owners of the datasets I used as I was given them for free.

PRACE SoHPCProject ID

1917

FMM: A GPU's Task

Noé Brucy

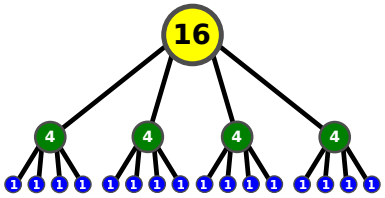


Is you graphics card able to run N-body simulations in a smart way? A complex tree algorithm, a sophisticated tasking system, is all that a task for a GPU? No, some will say, a graphics card can do only basic linear algebra operations. Well, maybe the hardware is capable of much more...

The Fast Multipole Method (FMM) is an algorithm to compute long-range forces within a set of particles. It enables to solve numerically the N-body problem with a linear complexity, where it would otherwise be quadratic. Doubling the number of particles only doubles the computation time instead of quadrupling it. However the FMM algorithm is hard to parallelize because of data dependencies. Tasking, meanings splitting the work into tasks and putting them into a queue helps a lot to give work to all threads. A working tasking framework for the FMM has been implemented on Central Processing Units (CPU) by D. Haensel (2018). Will such a tasking framework run efficiently on General Purpose Graphical Processing Units (GPGPUs or more simply GPUs)? The answer is not obvious at all, because CPU and GPU have a very different architectures. My goal this summer was to shed some light on that topic.

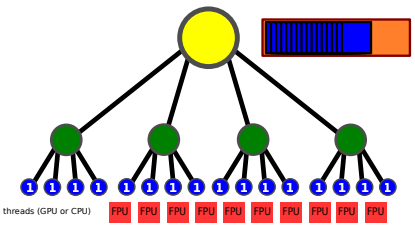
A smooth start

Let me first introduce the problem that we will use to test the tasking framework. It is a simplified version of one of the operators of the FMM, and we named it the *accumulation tree*.



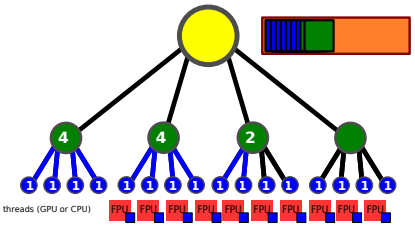
The principle is very simple: the content of each cell is added to its parent. So one task is exactly "adding the content of a cell to its parent". You already see that dependencies will appear since a node needs the result from all children before having its tasks launched. Imagine we have 10 processing units (GPU or CPU threads), the computation of the accumulation will be as follows.

Initialisation



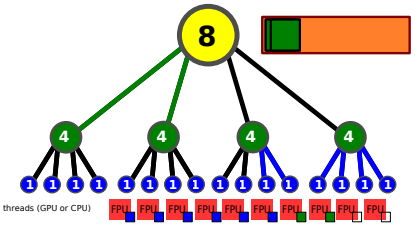
All leaves are initialised with 1. All tasks that are ready, that is all tasks from leaves, are pushed into the queue (blue tasks). Now we are ready to start.

Round 1



Ten blue tasks are done. Two new green tasks are ready; so they are pushed into the queue.

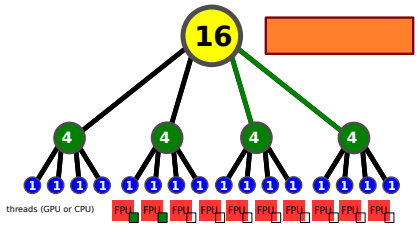
Round 2



The last six remaining blue tasks are done as well as two green tasks. The last two green tasks are ready and pushed into the queue. Here we can see that the tasking permits to maximise

the threads usage since all tasks that are ready can be executed.

Round 3



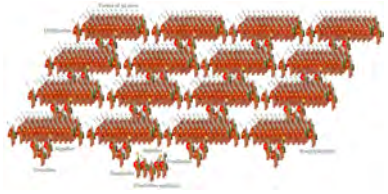
The last two green tasks are executed. We get the correct result, hip hip hooray!

And on a GPU?

As said in the introduction, such a tasking system works well on a CPU. Why can't we just copy-paste the code, translate some instructions and add some annotation to make it work on a GPU? Because many assumptions we can rely on CPUs do not hold anymore on GPUs. The biggest of them is thread independence. You can compare the CPU to a barbaric army: only few strong soldiers, any of them being able to act individually.

credits: Wikimedia CC-BY-SA

A graphics card however is more like the roman army, with a lot of men divided into unit. All soldiers within the same units are bound to do exactly the same thing (but on different targets).



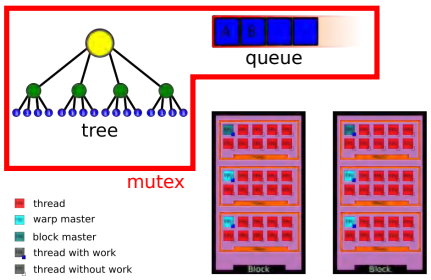
credits: https://patricklarkinthrillers.files.wordpress.com

Many of the problems caused by the special nature of the GPU were already tackled when I joined the project. Indeed, I was provided with a working version of the tasking framework applied to the accumulation tree example. However, it had serious performance issues, since the accumulation took 700 ms on an octree of depth 5. An octree is a tree where each node has 8 children, and an octree of depth 5 has 37449 nodes, which is not that much.

Time to speed things up!

In this section I will tell you my strategies to improve the performance. Let's begin by describing how the tasking framework I was provided with worked.

The starting point



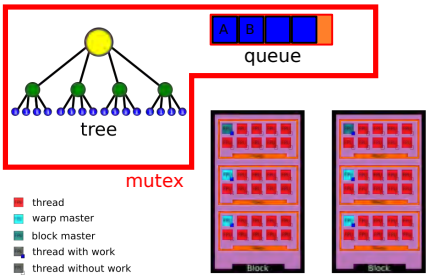
The protagonists of our story are the following. The threads are our roman soldiers. They are working on a Floating Point Processing Unit (FPU) and they do the computation. They are grouped in units of 32 threads called warps. Threads within the same warp should execute the same instruction at the same time, but on different data. Warps are themselves grouped into blocks. Threads within the same block can communicate fast through shared memory and they can synchronise together. However, no inter-block communication or synchronisation is allowed except through the global memory which is much slower.

The tree is another player. He lives in the global memory since all threads should be able to access it. At last thre is the queue, also living in the global memory. It is implemented by a chained list, meaning that its size can adapt to the number of tasks it contains. To avoid race conditions, access to the tree and the queue are protected with a mutual exclusion system (mutex). We rely on a specific implementation of a mutex for GPU. It allows only one thread to access the data protected by the mutex at a time. To avoid deadlocks (a kind of infinite loop when two threads wait for an event that will never occur), only one thread per warp tries to enter the mutex. We call it the warp master, and in this early version, only the warp masters can work, as follows:

1. Fetch a task from the queue
2. Synchronisation of the block
3. Execute the task
4. Synchronisation of the block
5. Push new tasks (if any)

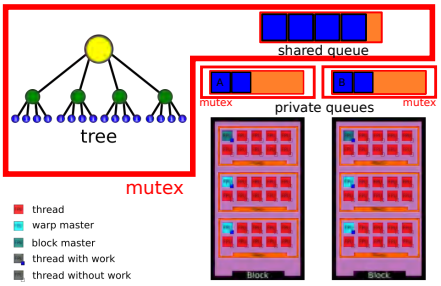
Step 5 is done only if the task done at step 3 resolves a dependency and creates a new task.

Cutting allocations



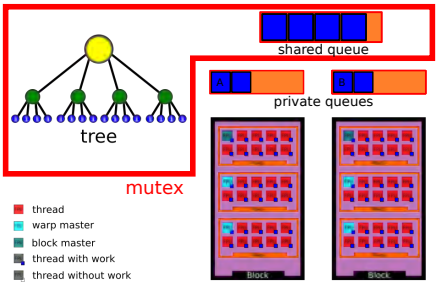
The first improvement was to change the queue implementation with my own relying on a fixed sized queue. The reason behind this is that memory allocation is very expensive on a GPU, and with adaptive size queue you are allocating and freeing memory each time a thread pushes and pops a task.

Reestablishing private property

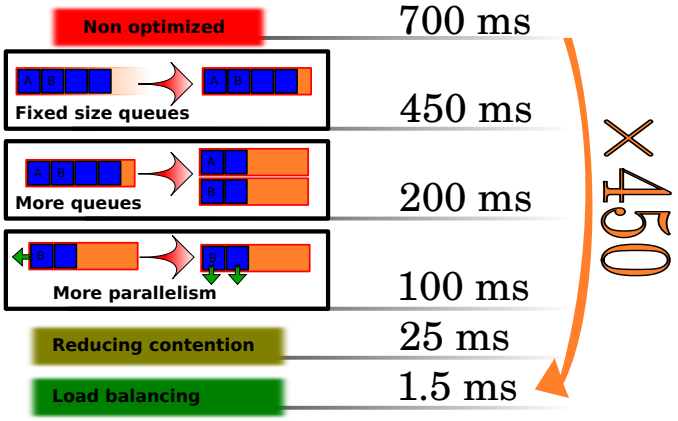


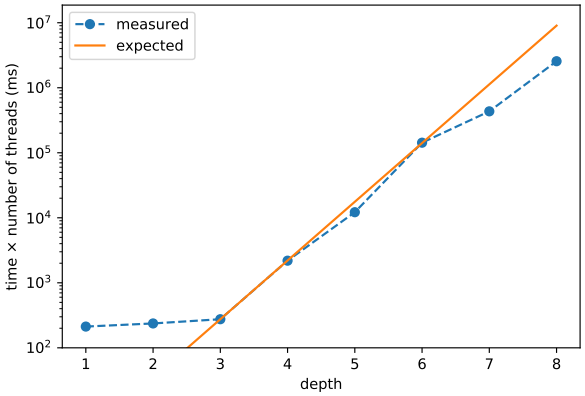
The second idea was to reduce the contention on the global queue since working threads are always trying to access it. I added small private queues for each block, that can be stored either in the cache or in the shared memory for fast access. The threads within a block use the block's private queue in priority, and the global queue as a fallback if the private one is full (push) or empty (pop).

Solving the unemployment crisis



For now only a few threads (the warp masters) are working. It's time to put an end to that. First, since threads within a block are synchronised, access to the private queue is done at the same time thus performed sequentially because of the mutex. I decided that only one thread per block will access the queue, and will be in charge to fetch

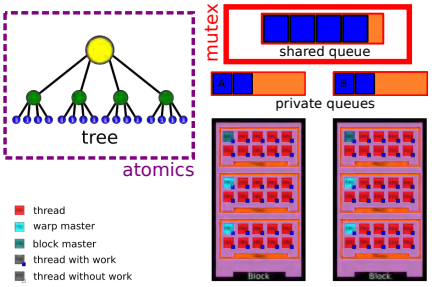




Left: approximate timings for the accumulation kernel for an octree of depth 5 on a Tesla P100 GPU, excluding initialisation.
Right: total thread execution time for several depths on the Tesla P100

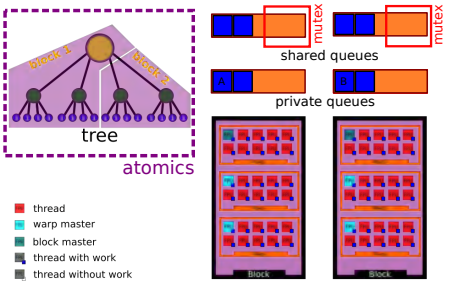
the work (step 1) and solve dependencies (step 5) for all the threads.

Breaking down the wall



Now that all threads are working, they are all waiting to enter the mutex protecting the tree, even if they are trying to access different parts of it. So I removed the mutex, and ensured that all operations on the tree are atomic. That's a bit like if there was a mutex on each node in the tree, making it possible to several threads to access the tree concurrently.

Fighting inequality



Removing the mutex from the tree resulted in a huge gain on time, so I tried to also get rid of it for the shared queue access. First I did split the queue so that there is one shared queue for each block. Because the queue is fixed in size, the operations of pushing and popping a task are independent. One block master can pop only from its own queues (private and shared) and can push to its own private queue, its own shared queue and also other block's shared queue. Thus, we do not need mutex protection for the popping operation.

Last but not least, the work must be equally shared between blocks (that is called *load balancing*). I provided a function that tells a calling thread in which shared queue a newly created task should be pushed.

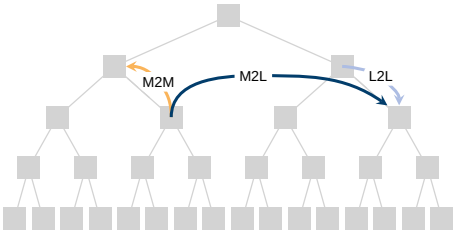
Was it worth it?

Now it's time to evaluate how much time we saved with our drastic politics. And the answer is: a lot! The top-left figure shows how much time the computation of the accumulation octree of depth 5 took for each of the above mentioned steps. The results are quite impressive, as my work reduced the execution time from 700 ms down to 1.5 ms, resulting in a speedup of approximately 450.

It also scales pretty well if we increase the size of the tree. Indeed one would expect that the total execution time (the measured time multiplied by the number of threads) increases eightfold if the depth is incremented by 1. That is because by adding one tree level, we increase the number of task approximately by a factor 8. The top-right graph shows that the scaling seems to be just as expected from the depth 3.

An important step to get this result was to parallelize the allocation and the initialisation of the tree, gaining a speedup of 900 for depth 5 and thus enabling runs with a depth up to 8. There is not enough memory on the GPU for higher depths. I also wrote a specific Python benchmarking programme to compare different versions of the code.

Towards a full GPU FMM solver



The accumulation tree example is very close to an operator of the FMM called M2M. There are two other tree operators in the FMM algorithm: M2L that compute interaction between nodes at the same level and L2L that is pretty like M2M but from the top of the tree to the bottom. I also implemented operators that have the same execution scheme as M2L and L2L. They work correctly and have execution times within the same order of magnitude as the accumulation tree. Therefore we are not very far from a taskified FMM algorithm running on GPU. The remaining work is the replacement of the simple addition in the tree by the the complex mathematical operations of the FMM.

PRACE SoHPCProject Title
Good-bye or Taskify!

PRACE SoHPCSite
Jülich Supercomputing Centre, Germany

PRACE SoHPCAuthor
Noé Brucy, France

PRACE SoHPCMentors
Laura Morgenstern, Jülich, Germany
Ivo Kabadshow, Jülich, Germany
Andreas Beckmann, Jülich, Germany


Noé Brucy

PRACE SoHPCContact
Brucy, Noe, AIM, CEA Saclay
E-mail: noe.brucy@cea.fr

PRACE SoHPCSoftware applied
Cuda, Nvidia profiler, Python.

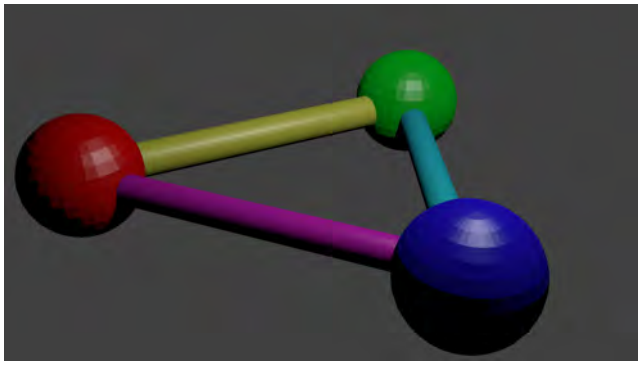
PRACE SoHPCAcknowledgement
Thank you very much Laura and Ivo, I really liked working with you!
credits: the GPU icon is adapted from Arthur Shlain (the Noun Project)

PRACE SoHPCProject ID
1918

High Performance Lattice Field Theory uses the computational power of supercomputers in order to produce a high accuracy lattice field simulation which cannot be created on conventional computers due to its computational expensiveness

High Performance Lattice Field Theory

Andreas Nikolaidis



The image above shows a subatomic particle composed by three quarks, it can be a neutron or a proton

Motivation of project: Understanding how the lattice simulation works in order to solve mass problem that occurred during the simulation for the propagation of a meson particle

Quantum chromodynamics (QCD). QCD theory is a quantum field theory that describes the strong interaction between quarks and gluons. Quarks are the subatomic particles composing the well known neutrons and protons and they occur in six flavours up, down, strange, charm, top and bottom. The proton for example is a composition of three quarks, 2 up quarks and 1 down quark. If the story ended here then we would be talking about quantum flavourdynamics. Due to the very small mass of the quarks they should have been constantly moving with speeds close that of light, so how are they bound together? At this point the strong force is introduced and it is called strong because it is indeed strong so that it can hold the quarks bounded together and it is the strongest of all four of the fundamental forces.

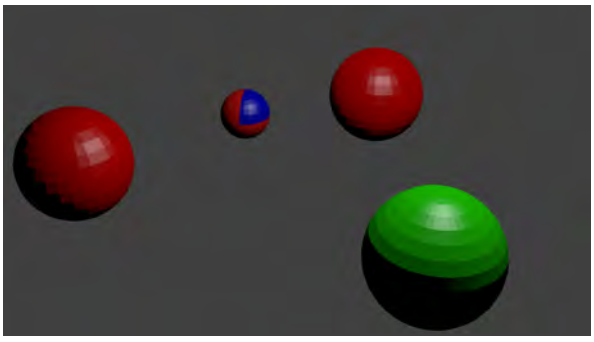


Figure 1: The small sphere is the gluon carrying an anti-red and blue colour. Once it reaches one of the big red spheres representing the quarks the anti-red with red will be annihilated and the quark will be blue

Carrier of the strong force is the gluon. Analogous to electric charge, in QCD, is the colour charge and instead of having positive and negative charge there are three colours representing it, red, green and blue (though colour has nothing to do with the real colours). Lattice QCD is a non-perturbative method for solving QCD problems, but why we use non-pertubative methods? The reason lies with the highly non-linear nature of strong force and the large coupling constant at low energies which makes it impossible for an analytical or perturbative solution to be found.

Problem

When there is a big project taking place the project breaks down to smaller parts and i was tackling one of the problems occurred during this project. The main project was lattice field simulation of how a meson (a subatomic particle composed by a quark and an anti-quark) propagates from a point X at a point X' and what concerns the simulation is what path the particle has used to get from X to X'. The data contains the number of measurements taken between these two points which is denoted by tmax (i.e. tmax= 28 means 28 measurements have been used). Now, while the simulation had been running just fine for two different data inputs the

results showed different rest mass for the same meson particle. So that tells us that there should be a point where there is an error which might occur by a variety of reasons i.e. mistake either at the data input or at the initial parameters added to the code or at the simulation.

Methods to tackle the problem

Firstly, out of the number of measurements taken that is mentioned above (tmax) some range of this data gives better confidence of results i.e. if tmax=28 instead of using the full range of measurements from 0-28 we get a range out of it, as an example from tmin= 5 to tmax=25 and observe what confidence, rest mass and error in mass this data provides. Now since we get different answers for different ranges it is reasonable that the results needed to be analysed in order to extrapolate the mass, error in mass and confidence for each of the tmin values and for different tmax values. This task was executed with the help of a bash script as follows. The script iterated over all tmin values for different fixed values of tmax and for each range the quantities mentioned above where extrapolated into a file.

Continuing, initial plots for the two data inputs were produced for mass vs tmin for a fixed tmax value where produced as shown in the figures 2 and 3 below.

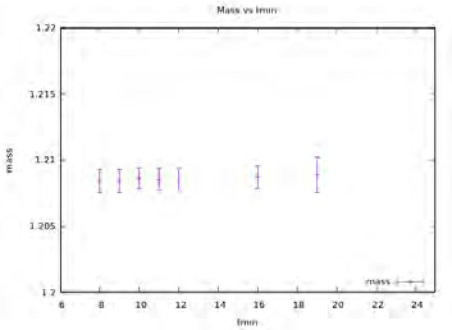


Figure 2: Initial plot of the smpt file which is one of the data inputs of the simulation code (tmax=28)

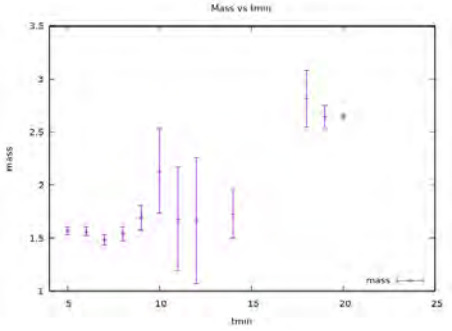


Figure 3: Initial plot of the smsm file which is the other input file of the simulation code (tmax=28)

Results and discussion

In order to understand where the problem occurred we can compare the two figures already mentioned. It is rather obvious from the two figures that figure 2 is at least to a certain degree consistent with the rest mass results while figure 3 shows very unstable results. Based on this deduction, the problem seemed to lie only with the one data input. So, by checking the input parameters for that file the error found to lie with the order that masses were fed to the fit-code, but what does that even mean? Well, the rest mass that fed the code was larger than the mass of the first excited state which cannot be the case since in the first excited state the particle holds more energy and thus it is heavier. After arranging that issue the above figures where re-plotted in order to see if the problem has at last been solved and this time with various values of tmax. The final and most important plot is one that shows the two results of the files on a single graph and the tmax values that were selected are the ones providing the greatest confidence. This is the figure 4 shown below

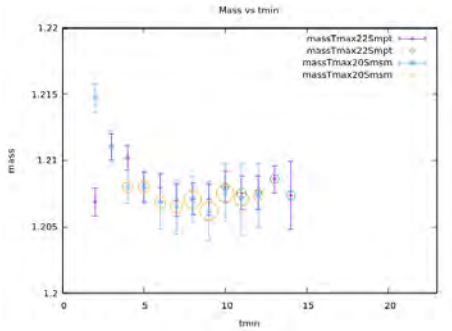
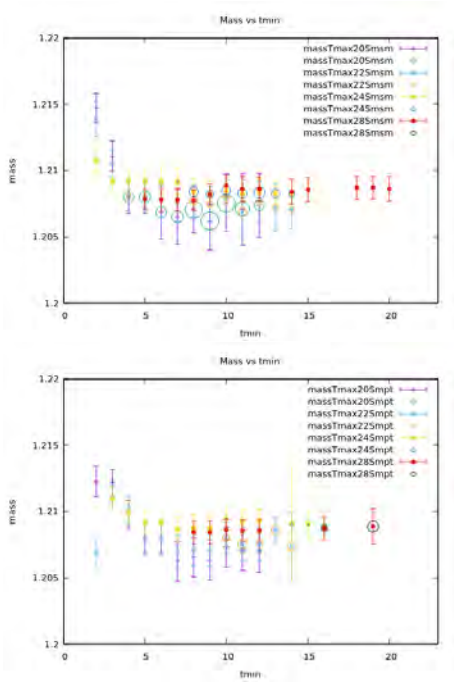


Figure 4: It should be noted that the circles shown on the graph symbolise the confidence of the data

So, by observation of figure 4 we can see that apart from some issues at the small values of tmin which are pretty reasonable due to the way that the simulation works, the data for both files is pretty stable and at the same range. We can thus conclude that the problem with the rest masses of the two input files has been solved.

Another two graphs were produced that include all the tmax values that were used to analyse the data.



Figures: The figures above were plotted in order to show conformance of the results after the mistake was solved.

Conclusion

The analysis of the data was successful and it showed directly where the errors. Though, at the beginning the issue was thought to exist with the range of values that it was used for the two files and at the end the error occurred due to the program not analysing the correct data. The error of the program was that it did not check whether the mass of the ground state was smaller than the one of the first excited state and that caused all the issues.

Acknowledgement

I wanna thank PRACE summer for this opportunity that was given to me and i would also like to thank my supervisors who helped me with learning new things, and to be honest almost everything was unknown to me.

PRACE SoHPCProject Title

Lattice QCD simulation

PRACE SoHPCSite

JSC (Julich supercomputer center), Germany

Andreas Nikolaidis

PRACE SoHPCAuthors

Andreas Nikolaidis, Cyprus

PRACE SoHPCMentors

Stefan Krieg, JSC, Germany
Eric Gregory, JSC, Germany

PRACE SoHPCContact

Andreas, Nikolaidis, Phone: +447871554308

PRACE SoHPCAcknowledgement

Thanks to my supervisors this summer i learned things that i wasn't aware before and i am really happy about it.

PRACE SoHPCProject ID

1919

Encrypting disks in HPC

Kara Moraw

Supercomputers which are publicly available have relaxed security measures by default. When it comes to processing sensitive data, further steps towards data security have to be taken. One possible measure is working on encrypted disks which, according to our findings, has an acceptable impact on performance.

There is an increasing demand for processing confidential data, such as the genetic makeup of humans, personal data, and data that, if leaked or modified, could have serious legal consequences. Normally, supercomputers and clusters are shared by many users, which makes it difficult to meet strict security requirements. Furthermore, network restrictions hinder the flexibility and the open character that make clusters popular.

PCOCC (Private Cloud On a Compute Cluster) is a promising technology that enables creating private HPC-clusters on existing clusters, by automating the setup of KVM virtual machines, connected with an overlay network. These guest private clusters can be tailored to the security requirements per project, whilst maintaining the flexibility and openness of the host cluster.

A security weakness in the current PCOCC software is that disk images that are used for storing persistent cluster data are not encrypted. Because the disk images reside on the host cluster file systems, confidential data could be read by users with enough privileges on the system. Since PCOCC is an open-source python-based tool, it could be modified to run virtual machines on encrypted disks.

Working in an HPC environment, performance is always a priority. So to evaluate the practicability and usefulness of adding the feature described above to PCOCC, we decided to run a series of benchmarks on a simple setup of KVM virtual machines with LUKS encrypted volumes.

LUKS encryption format

LUKS is short for *Linux Unified Key Setup* and provides a standard for encrypting disks in Linux operating systems. It takes care of the key management with a two-level key hierarchy: A strong master key is generated by the system and used to encrypt or decrypt the hard disk. The master key itself is encrypted by a user key and only this encrypted version is stored. To decrypt the hard disk, the user must provide his user key which is used to decrypt the master key which in turn decrypts the hard disk. This way, the user key can be changed frequently - this is fundamental for security - without re-encrypting the hard disk every time. Only the master key has to be re-encrypted which saves a lot of overhead.

LUKS makes use of a series of further security measurements to impede attackers calculating the keys, such as AF-splitter and PBKDF2 [REFERENCES]. The details are not relevant for this project and are therefore not discussed here.

It is possible to use different encryption algorithms, modes, key sizes and hash functions within LUKS. In this case, we limited our tests to the algorithms AES, Twofish, Serpent and CAST5 in the modes CBC, ECB and XTS. The following gives a short overview.

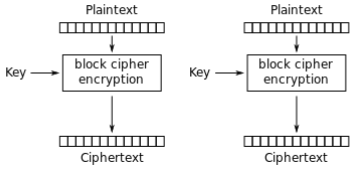
All four of the considered algorithms remain unbroken and are therefore secure. It is out of the scope of this report to explain their details, but the main thing to know in order to understand the significance of encryption modes is that they are all block ciphers. This means that they work with a key of a predetermined length and this key can only encrypt a message of the same length. For a short message, this might be alright, but when looking to encrypt a 500 GiB hard disk, a 500 GiB long key is hardly of use. Instead, the data is split

```
<secret ephemeral='no' private='yes'>
   <description>Example secret</description>
   <usage type='volume'>
      <volume>/guest_images/vol.img</volume>
   </usage>
</secret>
```

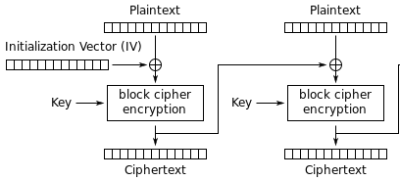
(a) XML file for secret object

```
<volume>
    <name>vol.img</name>
    <capacity unit="G">7</capacity>
    <target>
        <path>/guest_images/vol.img</path>
<format type='raw'/>
<encryption format='luks'>
<secret type='passphrase' uuid='dcb67'/>
<cipher name='aes' size='256' mode='ecb' hash='sha256'/>
</encryption>
    </target>
</volume>
```

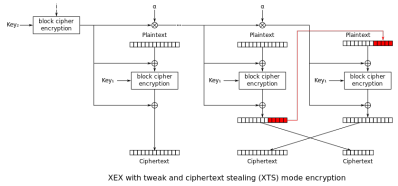
(b) XML file for volume



(c) Electronic Code Book mode



(d) Cipher Block Chaining mode



(e) XTS mode

Figure 1: On the left: XML files for configuration. On the right: different encryption modes.

into data blocks of the same length as the key (e.g. 256 bits). There are different possibilities to apply the encryption algorithm to those blocks, i.e. different *modes*:

ECB

The Electronic Code Book mode, depicted in 1c, is the simplest and fastest way to encrypt blocks of plaintext. Every block is encrypted separately with the chosen algorithm using the key provided. This can be done in parallel. However, it is highly deterministic: identical plaintexts have identical ciphertexts.

CBC

The Cipher Block Chaining mode, seen in 1d, is less deterministic, but slower. The encryption is randomised by using an initialisation vector to encrypt the first block. Every subsequent block's encryption depends on the block before, i.e. also on the initialisation vector. This cannot be done in parallel.

XTS

The XEX-based tweaked-codebook mode with ciphertext stealing, seen in 1e, uses two keys and supports encryption of sectors which cannot be divided into equal-length blocks.

Test environment

PCOCC itself creates a set of virtual machines working together as a cluster, with KVM as their hypervisor. Our test setup lies on the HPC-Cloud [REFERENCE], a service offered by SURFsara. It lets users create virtual machines running on SSDs. One test instance consists of two virtual machines on the HPC-Cloud: One acts as the host for the virtual machine to be benchmarked, and one acts as an NFS-Server which will contain the encrypted disk.

Within the virtual machine acting as a host (we will be calling it "host VM" from now on), we recreated what PCOCC does in a simpler way by just defining one virtual machine instead of a cluster. Details on doing this can be found in the REDHAT DOCUMENTATION, but the essential steps for creating the encrypted virtual volume the machine should run on are:

1. Define a secret.
2. Define encryption to be used.

Defining a secret

With libvirt, an object of type "secret" can be created from an XML file like the one in 1a.

Attributes *ephemeral* and *private* describe that the secret is only kept in memory as opposed to being stored persistently and that the value to be associated is never revealed. This definition also declares which volume is to be protected by the secret. Note that this definition does not contain the actual passphrase. It has to be set afterwards by calling `virsh secret-set-value`.

Defining the encryption

Telling libvirt which encryption to use is part of the volume definition. Again, a new encrypted virtual volume can be created from an XML file like the on in 1b.

Here, format, secret and cipher to be used are declared. The *secret* object points to the secret created above. There are different encryption algorithms, modes, key sizes and even hashes available.

Benchmarks

To evaluate the performance, we set up eight differently encrypted volumes and had a virtual machine transcode a 17-minutes long video on each of them. The configurations we considered are listed in table 1. The results displayed in 2 and 3 show the average time and the standard deviation after five runs on each configuration.

The results in 2 show that there is

no significant impact on the user time. The average performance is decreased by $0.2 - 3\%$. There is a higher impact on the system time as seen in 3. The performance is decreased by $13 - 30\%$. Due to the small number of runs, it is not possible to make a general statement on which algorithm and mode have the lowest or highest impact on performance. Nevertheless, the results indicate that the details of the configuration affect the performance.

Table 1: Configuration of volumes

Volume No.	Cipher	Mode	Key Size	Hash
1	None	None	None	None
2	AES	ECB	256	SHA256
3	AES	XTS	256	SHA256
4	AES	CBC	256	SHA256
5	AES	CBC	128	SHA256
6	Twofish	CBC	256	SHA256
7	Serpent	CBC	256	SHA256
8	CAST5	CBC	128	SHA256

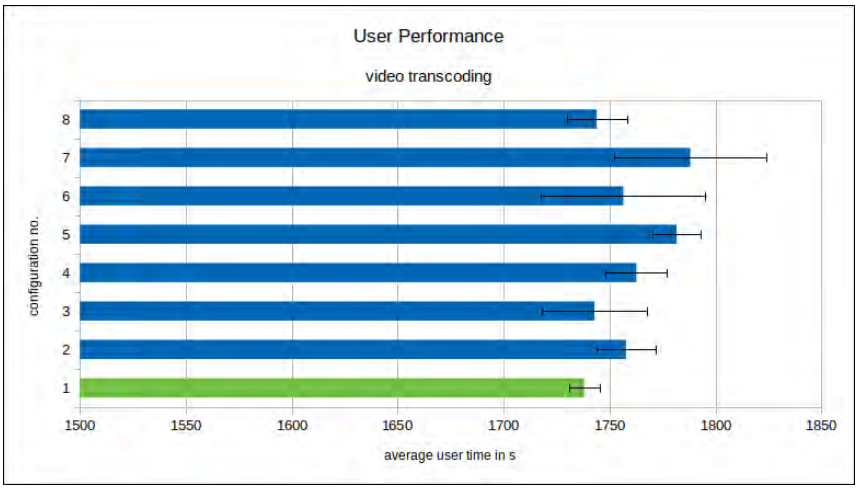


Figure 2: Average user time when using encrypted disks (blue) compared to a non-encrypted disk (green).

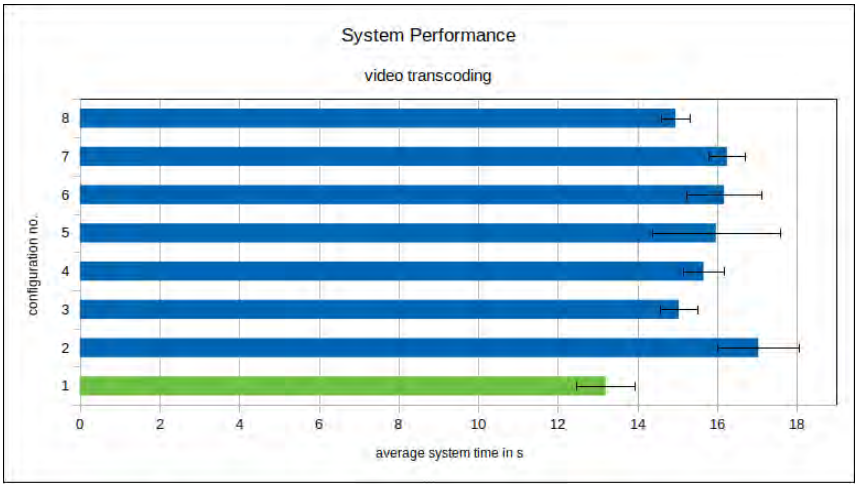


Figure 3: Average system time when using encrypted disks (blue) compared to a non-encrypted disk (green).

Discussion

Looking back at the process, we find it was fairly simple to set up an encrypted disk for a virtual machine. While we could not verify the configuration within PCOCC itself due to the time constraints, it looks like encrypted disks could be added in after minor changes to the PCOCC code. The benchmark results show that the encryption does impact the performance, but only to a moderate extent. Also, they suggest that performance demands can be balanced with security demands since different configurations of the encryption lead to different timings. Further work could research the impact of encryption modes or algorithms on overall performance and work towards the incorporation of encrypted disks in the PCOCC code.

PRACE SoHPC Project Title

Encrypted volumes for PCOCC private clusters

PRACE SoHPC Site

SURFsara, Netherlands

PRACE SoHPC Authors

Kara Moraw, Universität Bonn, Germany

PRACE SoHPC Mentors

Lykle Voort, SURFsara, Netherlands
Caspar Van Leeuwen, SURFsara, Netherlands


Kara Moraw

PRACE SoHPC Contact

Kara Moraw, Universität Bonn
E-mail: k.moraw@uni-bonn.de

PRACE SoHPC Software applied

KVM, PCOCC, libvirt

PRACE SoHPC More Information

KVM: https://www.linux-kvm.org
PCOCC: https://github.com/cea-hpc/pcocc
libvirt: https://libvirt.org/

PRACE SoHPC Acknowledgement

I would like to thank my mentors Lykle and Caspar for organising this project and offering the opportunity to get to know SURFsara. My colleagues at SURFsara also deserve a big thank you for listening to my problems over lunch and offering their expertise. Also, I would like to thank EPCC's Amy Krause and Nick Brown for agreeing to have me as an intern two years ago and introducing me to the HPC world. And last but not least, thanks to Allison Walker for always listening and being the best flatmate you could hope for.

PRACE SoHPC Project ID

1920

A data pipeline for weather data

Allison Walker

For ecologists studying bird movement, new data sources can contribute a deeper understanding of migratory patterns. This project focused on building a data pipeline enabling ecologists to efficiently access and visualise a new stream of meteorological radar data.

Birds have amazing capabilities for migration, with some species capable of travelling vast distances each year. While ecologists have traditionally focused on GPS tracking when studying migratory patterns, it is increasingly important to integrate additional data sources. In particular, ecologists at the University of Amsterdam are beginning to work with a new stream of weather radar data to track flocks of birds, and answer questions like: how and when are birds most active? How does weather impact when, where and how birds migrate?

These questions are important to answer for various reasons. Migrating birds share airspace with airplanes, windmills, and other man-made structures like high-rise buildings. It's important for both the birds' safety and the longevity of these aircraft and structures that we are able to accurately predict when flocks will be migrating and where. Some of the real-world implications of accurately predicting bird migration are:

- The aviation industry as well as Air Forces across the globe adjusting the scheduling of flights.
- Wind farms stopping or slowing their wind turbines when a large number of birds are flying through.
- Switching off lights in tall buildings and communication towers, which have been known to confuse birds flying at night.

Figure 1: A falcon with GPS tracker, the traditional means for capturing data on bird migration.

Project Overview

The aim of this project was to improve the accessibility of this weather data so that ecologists can more readily study weather patterns and local bird migration for a given radar.

Where the researchers are used to working with relational databases, this project explored the suitability of replacing these with static storage along with an accompanying working environment and practices that would form a virtual lab. The intention was that querying this static database still feels natural, with ecologists able to apply their existing IT knowledge. This project was also intended as an investigative analysis into the potential infrastructures for data sources beyond weather and GPS. A set of scientific workflows and a guide for best practice for storing and working the data stream will provide a foundation for future solution design. For the purposes of this internship, however, the use case was meteorological weather data.

The solution developed allows ecol-

ogists to select the metric, the time period, the elevation angle and the visualisation format desired for the output. The final visualisation is efficiently generated, giving researchers more time to focus on ecology.

Key tools used in developing this pipeline include: Parquet storage format for efficient filtering and loading, Spark for distributed computation, Python as the key programming language, Wradlib, Geoviews and Holoviews libraries for visualisation of the transformed data. The following sections of this paper will explain in greater detail the application of these tools.

Learning the Domain

The domain specificity of this project required a deep-dive into the world of radar. A critical first step in this project was understanding the way in which radars capture and store data, the metadata collected at each sweep, and the prevailing methodologies for reading and visualising these data.

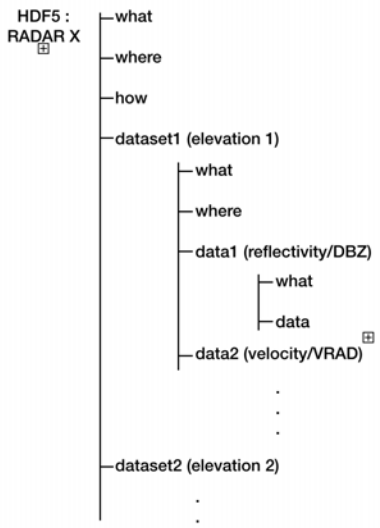


Figure 2: HDF5 file structure for radar data.

Radar collects information about objects in the surrounding area by using radio waves. The returned echoes can be used to identify aircraft, weather patterns, and even flocks of birds. Every time a radar completes a sweep, a significant amount of metadata is captured and stored. Examples of this metadata are: radar latitude, radar longitude, radar height, sweep elevation angle, sweep range, and timestamp. In terms of the actual data, information is captured for up to 16 different metrics in each sweep and stored into an array of dimensions (azimuths, radius). All of these data and metadata need to be considered in order to correctly project the data captured into an accurate visualisation.

A typical radar might perform a fresh sweep every 15 minutes, meaning 96 sweeps per day. Data collected in each sweep is stored in its own file, in an HDF5 format. HDF5s are hierarchical files, and somewhat complicated to navigate. The highest level is the elevation angle, and nested within each angle is further nested information about the different metrics and their associated metadata. In the prevailing approach, the most time-consuming step in processing these files was filtering through the multiple layers and metadata to retrieve only the data arrays for the elevation angle and metric of importance.

Spark and Parquet

The many complexities of the radar measurement system meant that it was important that the solution built could easily filter and return only the data that the researcher is interested in for their particular question. Parquet was the most sensible format for these purposes. Parquet is a column-oriented data storage format that is efficiently filtered, and highly compatible with Spark, the next key tool in this project's toolbox.

```
root
 |-- elangle: double (nullable = true)
 |-- year: long (nullable = true)
 |-- month: long (nullable = true)
 |-- day: long (nullable = true)
 |-- hour: long (nullable = true)
 |-- minute: long (nullable = true)
 |-- second: long (nullable = true)
 |-- quantity: string (nullable = true)
 |-- data: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- shape: struct (nullable = true)
 |    |-- _1: long (nullable = true)
 |    |-- _2: long (nullable = true)
 |-- height: double (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
```

Figure 3: Parquet fields.

Therefore, the critical first step in building the data pipeline was to convert HDF5s to Parquet for a year's worth of data from a handful of European radars. This process required an understanding of which metadata would be required to accurately visualize the data, to ensure that all relevant information was transferred from HDF5 to Parquet. The structure of these Parquet files can be seen in figure 3.

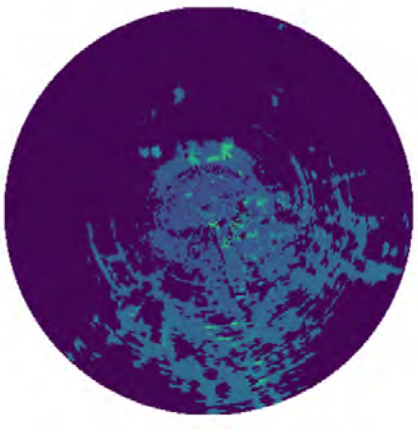


Figure 4: A PPI visualisation from the Wradlib library.

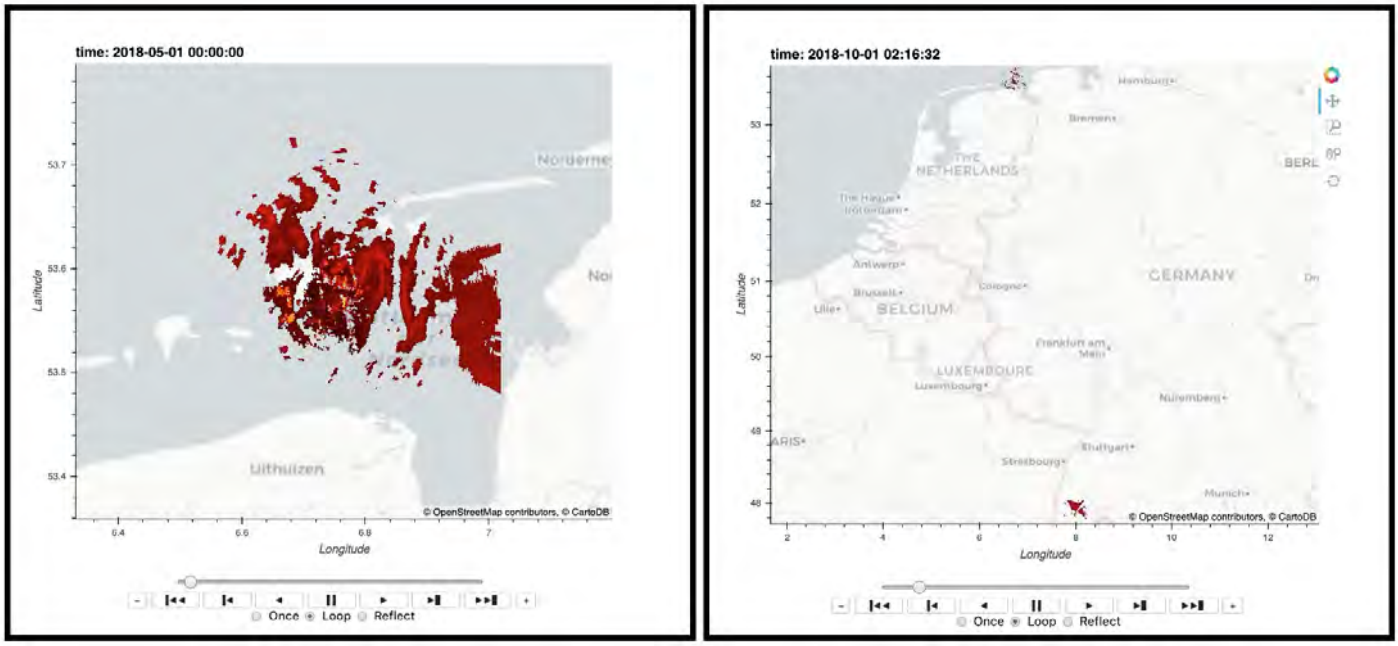
With the radar data formatted into Parquet and saved in Minio - the object storage software used in this project - the next step in the data pipeline was to load the relevant data. Spark - a general-purpose cluster-computing framework - was critical in this step. Functions for filtering and loading data were written in pyspark, and run in a Kubernetes cluster. This cluster has 25 machines, each with 4 nodes. Already by this stage the efficiencies of the new infrastructure were clear. Where 200 seconds are needed to filter, load and perform a groupby function over relevant data from a series of HDF5 files, only 120 seconds were required for the same process from Parquet.

Radar Visualisation

Transforming a data array and metadata into simple visualisations came next. Wradlib, a python library for Weather Radar Data Processing, was very important to understand the conventions used for transforming polar data (where the dimensions are azimuths and radius) into gridded data that can be more easily visualised. Again with the help of Spark, the Wradlib library was used to produce visualisations as seen in figure 4. These were further expanded into time series visualisations in a gif format (which unfortunately cannot be demonstrated in this article format).

Georeferencing

While these visualisations are useful, ecologists also require information about the geographic location of these weather patterns. In order to maximise the usability of the radar data, it was therefore important to add a geographical representation to the visualisations.



Key tools used in this stage of the pipeline were: Geoviews, Holoviews and Xarray. These are all python libraries that allow us to place the visualised radar data exactly where it belongs on the map.

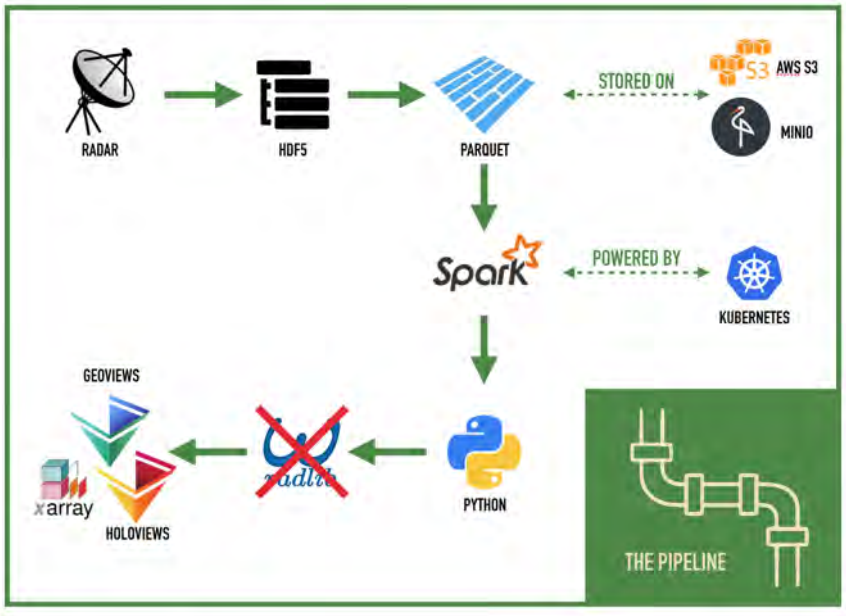
Geoviews and Holoviews expect geographical data to be in an Xarray format. These are essentially Numpy arrays, but with the addition of dimensions that summarise the geography and time of each datapoint. Utilising the principles from Wradlib, the raw data arrays were transformed into a gridded format, and converted to Xarray. In fact, this step removed the reliance on the Wradlib library altogether.

Finally, the Xarray datasets were converted into Geoviews Images and then projected onto a Cartolight map.

The final visualisation, with zoom and time-slide interactivity is shown in the figure above. Geoviews is capable of projecting information for multiple radars in the one view, so this solution is easily scalable for multiple radars, once that data is also transformed to Parquet format.

Final Product

The final output is, as intended, a functional data pipeline allowing ecologists to query and visualise radar data based on their desired filters. Though the exact timings of the previous approach are unconfirmed, it is clear that the solution developed cuts hours of time from the process to develop these weather visualisations.



Future Work

A number of further optimisations remain. Firstly, the role of colour in radar visualisations cannot be underestimated. The current colourmap is not consistent with what the researchers are familiar with, so needs to be changed. Next, the ecologists often chop out data from the furthest scans due to loss of accuracy, but all data has been included in the current solution. Next, there is a need to optimise the structure of the Parquet files and bring in additional metadata from the HDF5s. Finally, this data pipeline can be adapted to produce vertical profiles; a separate visualisation derived from the same HDF5 files.

PRACE SoHPCProject Title

Large scale date techniques for research in ecology

PRACE SoHPCSite

SURFsara, Netherlands

PRACE SoHPCAuthors

Allison Walker, Australia


Allison Walker

PRACE SoHPCMentor

Matthijs Moed , SURFsara, Netherlands

PRACE SoHPCContact

Allison Walker
E-mail: allison.walker@bts.tech

PRACE SoHPCSoftware applied

Spark, Python, Geoviews, Holoviews, Parquet

PRACE SoHPCAcknowledgement

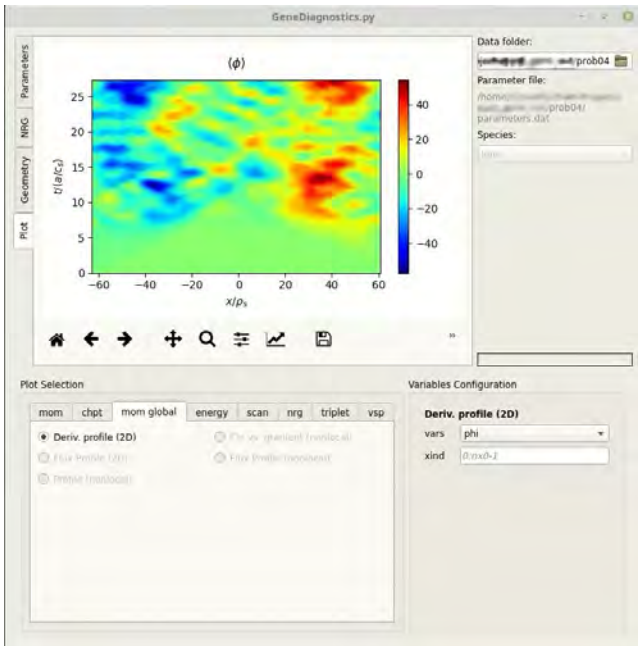
I would like to acknowledge the invaluable support of the project mentors Matthijs Moed and Ander Astudillo. I learnt even more than I expected, and owe my learning and productivity to both Matthijs and Ander and their constant support. Further, the summer in Amsterdam would not have been so successful and fun without the friendship of Kara Moraw.

PRACE SoHPCProject ID

1921

Visualising the enormous amount of data from plasma fusion simulation, by creating an open-source visualization software.

A glimpse into plasma fusion



Arsenios Chatzigeorgiou

Plasma fusion simulations have the potential to guide a minimisation of the plasma microturbulences and make plasma fusion viable. GENE is an open-source code that performs such simulations. The output, however, is visualised with a proprietary-IDL software. In this project, we reproduced the IDL plotting schemes, using open-source alternatives.

Plasma fusion is the holy grail of the energy science. By mimicking the sun's procedure, a nuclear fusion reactor can produce enormous amounts of energy with no significant waste and using only isotopes of hydrogen as substance (^2H and ^3H). In order to produce such energy however, very high plasma temperatures are needed.

Research on plasma fusion for energy generation started back in 1940. However, until today, a functional reactor able to generate power have not become available. The major problem is the inability to create steady state plasma fusion. This means that plasma is kept stable and at high temperatures for long periods of time during which the energy is being constantly emitted". Until now the plasma process duration is being measured in milliseconds and getting plasma in steady state for just a few seconds is challenging.

One of the reasons the plasma doesn't reach steady state, is the plasma microturbulences. It is what happens when you mix plasma of different temperatures together. Cold plasma absorb heat from hot plasma, and the average temperature is lower than desired.

Research on plasma fusion reactors is also very time-consuming, expensive and a lot of resources (human and material) are needed to build a one. A reactor that could potentially generate enough energy has bigger size than a middle shaped human, making the creation even more expensive. Therefore a valid prediction of the reactor's function is crucial, essential and mandatory.

The scientific problem

In order to foresee and avoid plasma microturbulences, plasma behavior is being investigated through plasma modeling or plasma fusion simulations.

Scientists try to find the ideal reactor's characteristics (the geometry of the reactor, the properties of the magnets that will speed plasma etc), that minimises plasma microturbulences and therefore maximise the energy produced.

One of such plasma simulation utilities is GENE, a **G**yrokinetic **E**lectromagnetic **N**umerical **E**xperiment, an open-source code that can simulate among other things, the plasma microturbulences detected in reactors. These simulations demand big computational power, and may only run in HPC clusters. GENE, uses MPI (Message Passing Interface) for the parallelized part of code.

The GENE algorithm, generates some numerical ASCII files but mostly big, complex binary files as an output, but no visualisation occurs.

Current GENE status

In the GENE suite, the option offered for analysing and visualising the results is an IDL plugin. This plug-in uses the output files and produces a large list of plots (around 40 different plots) that can be accessed via the IDL GENE diagnostics GUI.

IDL however, is not publicly available. In order to fully analyse and visualise the results, a licence is required. The user interface, is also very old-fashioned, and a lot of configuration needed makes the GUI less user-friendly.

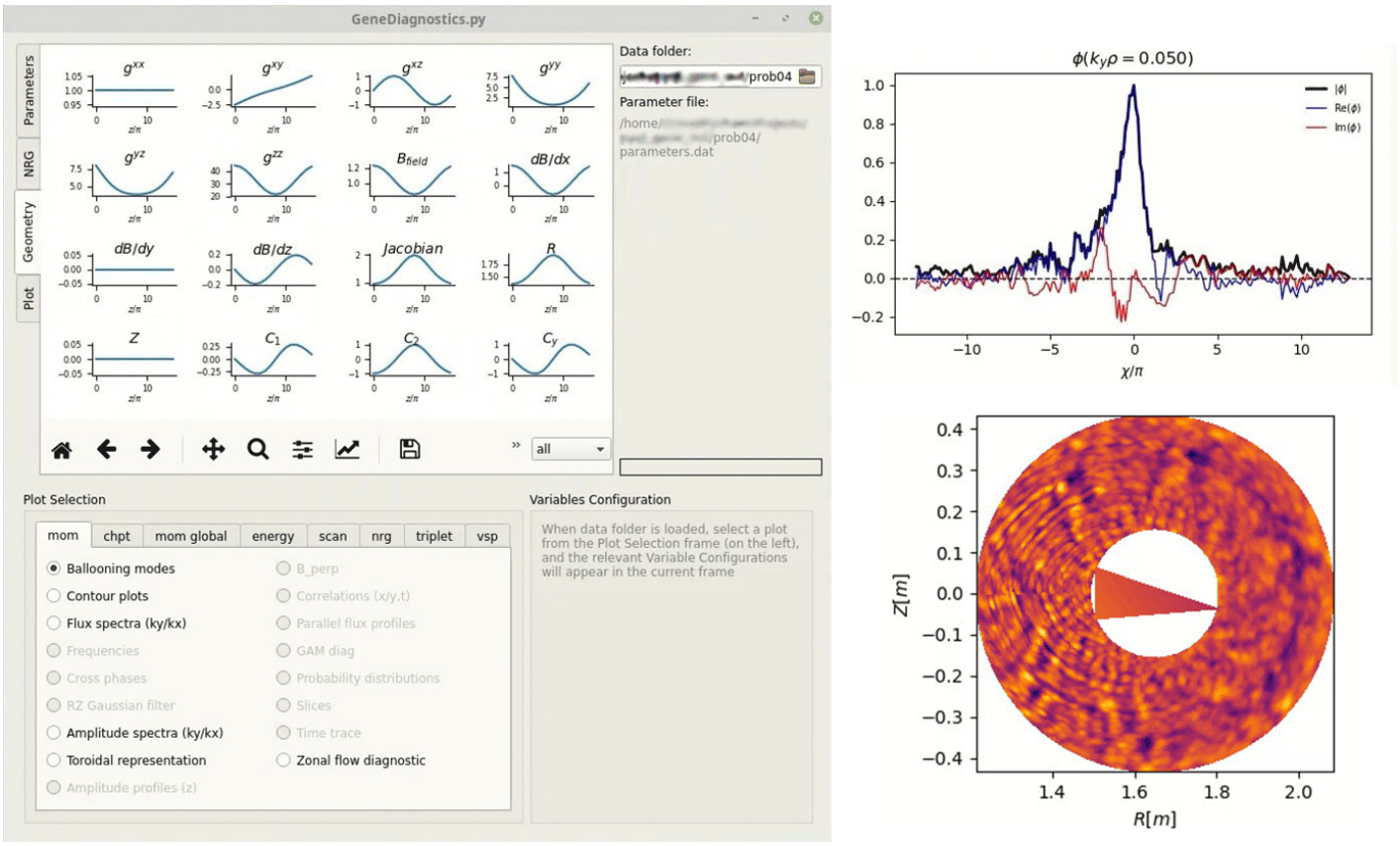


Figure 1: On the left, the our GUI is presented, with selected plot Geometry which stands for geometric elements of the reactor. Top right plot is balloning modes, and bottom right is toroidal representation.

Our accomplishments

In this project we managed to reproduce some of the IDL plotting schemes, in a more simple, modern and open-source GUI. The code was written in Python 3.7 and PyQt5 was used.

PyQt5 is a Python Toolkit based on the C++ Qt5 application framework. It utilises widgets as graphical interface elements, and provides the ability to create layouts of GUIs (Graphical User Interface). For plotting the data we employed the **matplotlib** library and numpy was used as well. For some plots, we used some scripts from a python implementation that existed in the GENE git repository, but was not currently in development.

One of the biggest difficulties we came up to, was that we didn't know enough about the GENE output files. Initially, we weren't able to access the **binary files** of the results. Also, formulas that produced the plots were not mentioned and the number of different variables surpassed the greek alphabet symbolism. To make matters worse, the IDL scripts were also poorly documented.

The *deux-ex-machina* was the aforementioned python scripts. Those scripts, were able to **read** the files, **manipulate** the data, and **draw** many of the IDL plots, but there was not GUI. Despite the bugs, the chaotic and undocumented way there were written, and some incompatibilities, we managed to use them and provide some valid plots and animations from the GENE output files. Animations that weren't available in the IDL plug-in since the GUI didn't support live video or animation representation.

To be fair, the number of different plots we finally provide is not big enough. We managed to produce 13 out of 40 plots, but we didn't have enough time to reproduce all of them. Also, since the GENE IDL plug-in is a proprietary program, we also found difficulties about running **benchmark cases**. The native HPC in University of Ljubljana didn't have an IDL license and GENE or IDL were not installed in the supercomputer. Thankfully, my mentor had access to the MARCONI HPC cluster, in Bologna, where GENE and IDL are installed and licensed, and we were able to run a few benchmark cases.

Initially, apart from the GUI implementation, we planned to create a manual to help the user navigate in our program. But a different approach prevailed. What if we made the GUI so ridiculously simple, that the manual would be redundant? So, we made everything automated, and the user has to select only the GENE results folder and choose a plot to visualise. The simple instructions needed are printed in the GUI.

Don't get me wrong, the reproduction of IDL is far from done. Re-implementation of the GENE Python scripts, enrichment of the plotting schemes available, and different benchmark cases are some of the things need to be done. However, solid foundation for an open-source visualisation of gyrokinetic data has been laid.

PRACE SoHPCProject Title
Visualization schema for HPC gyrokinetic data

PRACE SoHPCSite
University of Ljubljana, Slovenia

PRACE SoHPCAuthors
Arsenios Chatzigeorgiou, [UoA] Greece

PRACE SoHPCMentor
Dejan Penko, UL, Slovenia


Arsenios Chatzigeorgiou

PRACE SoHPCSoftware applied
PyQt5, GENE, IDL

PRACE SoHPCAcknowledgement
I have to thank my mentor Dejan Penko, and his colleague Gregor Simic. Without their help I wouldn't be able to reach at any point close to those results.

PRACE SoHPCProject ID
1922

Predicting Electricity Consumption

Khyati Sethia

Global electrical energy consumption is increasing rapidly. In order to make accurate electricity buy for selected time interval and enable smart planning, a predictive model is built for the consumption of end-user electricity.

The consumption for electrical energy is increasing rapidly. The selected Slovene company which sells electricity to its customers wants to build a customer electricity forecasting system in order to make better spending forecasts from the consumer's consumption history and data on influential related factors, thereby making it more accurate to know how much electricity should be bought for the selected time interval, which will ultimately save energy and, increase the profits for the company and decreases costs for the end-users.

Here, different influential external factors like geography-wise weather, holidays and time are examined and study is done to find the correlations for the prediction of energy consumption over time. Process is developed for predicting consumption, which enables smart ordering and planning of energy consumption and thus great savings. Algorithms for predicting electricity are developed in the R environment and then adapted to work over big data databases and NoSQL MongoDB databases. This will serve as a basis for developing a new billing system for end consumers. The following figure shows the project flow schema:

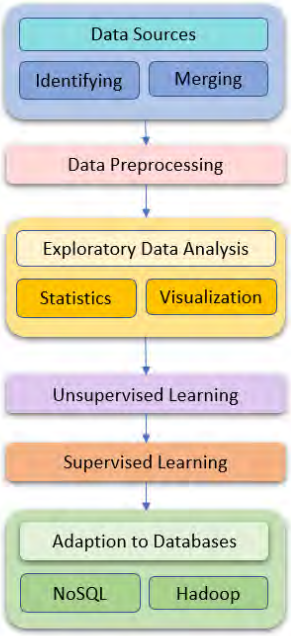


Figure 1: Process Flow

Data Sources

The dataset is a 15 minutes Electricity Consumption data spanning one year for 85 end-users. Fields are:

1. Date of Consumption
2. Time of Consumption
3. Region ID
4. Consumer ID
5. Consumption in KW

After the identification of data and their sources; the influential external factors for the consumption of end-user electricity are investigated. The focus is on the calendar and the weather. For this, the consumption data is fused with data about influential external factors and exploratory analysis is performed to understand how the selected factors influence the energy consumption. The weather data is downloaded from the Environmental Agency of the Republic of Slovenia (ARSO) website. The holidays in Slovenia information is obtained from the Time and Date Slovenia weblink.

Data Preprocessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format.

- The files are extracted from the nested zipped folder (received as input data from the Slovene Company) and bind-ed together into one file.
- The column formats are changed: European number format is converted to US number format, the date format is changed.
- Date related columns are derived: Week number, day of week, parts of the day, etc.
- Similar data processing as above is done for external factors - Weather & Holiday datasets.
- Special characters are replaced in Region names.
- Weather columns are normalised.
- All the three datasets are then merged.
- Day Type information is derived whether it is a Holiday, Weekend or a Working day.
- The long holiday information is derived i.e. suppose Thursday is a holiday then it is highly likely that most of the people take Friday also as a leave day, thus making it a long holiday. A similar case is considered for Tuesday.

Data Analysis

Data analysis is the process of evaluating data using analytical and statistical tools to discover useful information and aid in business decision making. We perform data analysis using two methods - Data Mining and Data Visualisation.

Total Consumption Analysis: Consumer 33 has maximum consumption compared to other consumers.
Day Analysis: Consumer 33 has maximum consumption for all day types; with 65% on working days and 29% on weekends. But the Consumer 35 has the highest overall 31% consumption on weekends and Consumer 7 has highest 94% consumption on Working days. Majority of consumers (except few) have high consumption on weekdays and less on weekends.
Period Analysis: We find that 67% of the consumers have highest consumption during noon while 29% of consumers have highest consumption in night. No consumer has highest consumption during evening.

In Figure 2, the consumption varies with temperature; more electricity is required at very low and very high temperatures.

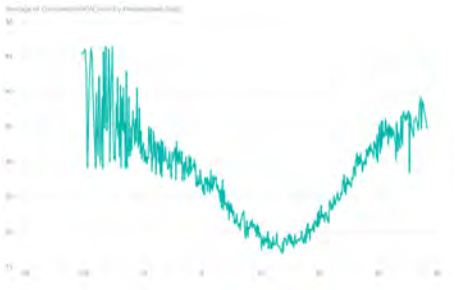


Figure 2: Variation of Consumption with Temperature

Figure 3 shows the correlation of various weather factors with consumption. Here the temperature is scaled by subtracting 10 and taking the absolute value. It can be seen that Consumption has high correlation with (scaled) temperature and radiation.

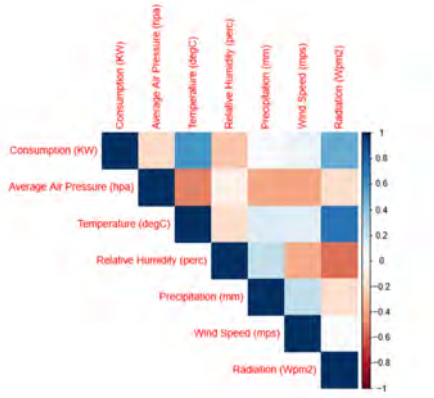


Figure 3: Correlation of various weather factors with consumption

Though, there are many consumers who don't exhibit similar temperature vs consumption behaviour.

In Figure 4, it is shown how the consumption varies in a week day (every 15 mins, hence a total of 96 data points in a day). The consumption is low during the night and high in the day with peak in the early morning. It has also been observed that the consumption decreases significantly on weekends.

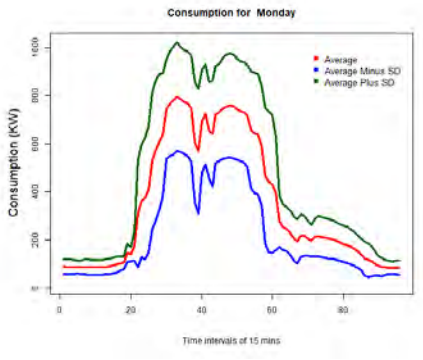


Figure 4: Electricity Consumption by Day

Unsupervised Learning

Clustering is a Machine Learning technique that involves the grouping of similar data points while data points in different groups should have highly dissimilar properties.
In this project we perform Hierarchical and K-Means Clustering.
Cluster analysis is performed to generate clusters of days with similar kind of electricity consumption with Euclidean Distance and other custom distance metrics. The euclidean distance among data points is calculated using the following formula:

$$EuclideanDistance = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \quad (1)$$

We tried these methods to find the optimal number of clusters for all consumers - Elbow Method, Gap Statistics and Hartigan. The method that gave the least prediction error is Hartigan index. The Hartigan index is computed using following equation:

$$hartigan = \left(\frac{trace\,(W_q)}{trace\,(W_q + 1)} - 1 \right) (n - q - 1) \quad (2)$$

Where
$W_q = \sum_{k=1}^{q} \sum_{i \in C_k} (x_i - \bar{x}_k)(x_i - \bar{x}_k)^T$ is the within-group dispersion matrix for data clustered into q clusters, q $\in$ (1, ..., n-2).
x_i = p-dimensional vector of objects of the i^{th} object in cluster C_k,
x_k = centroid of cluster k,
n is the number of observations in the data matrix.

In Figure 5, we have obtained these highlighted groups or clusters for one

consumer using statistical methods. Here the dendrogram visualize his daily consumption which is divided into three different groups of days.

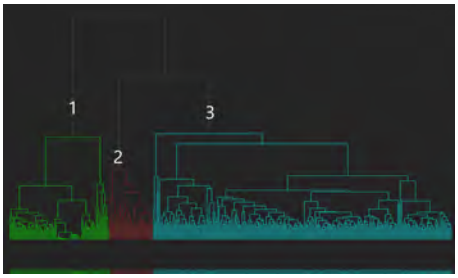


Figure 5: This dendrogram shows various clusters for one customer.

Figure 6 below, shows the clusters obtained using the K-Means Clustering for one consumer. This graph explain the data point variability using PCA (the method that minimises the error orthogonal (perpendicular) to the model line).

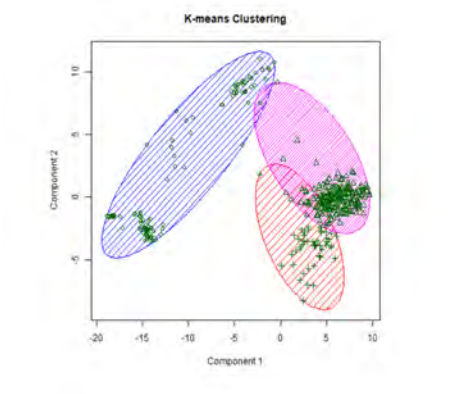


Figure 6: This clusplot shows various clusters for one customer.

The clusters are formed with similar consumption pattern in same cluster and very different from other clusters. We also studied that relationship exists between the clusters with day of week & holidays

Cluster Data Prediction

Using this cluster information, the relation among various factors like day of the week and the holidays is found. We get better results using K-Means Clustering than Hierarchical Clustering. For now, a process has been developed which uses this cluster information to predict consumption. To measure the prediction accuracy, the error percentage is calculated between the actual and predicted consumption. This error is equal to the difference in area between the two curves and is given by below equation and represented using Figure 7:

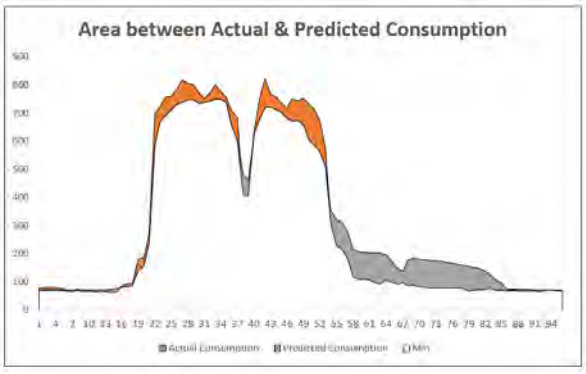


Figure 7: Area under the Actual and Predicted Curve

Adaption to NoSQL Database

NoSQL MongoDB is used to store the dataset for enhanced scalability in terms of storing large volumes of data, for flexibility with JSON-like documents and for faster retrieval, ad-hoc queries, indexing, replication and MapReduce programs. R 'Mongolite' package is used to work with NoSQL MongoDB Database. Using this package following was performed: data retrieval, manipulation, and analysis using the data stored in MongoDB.

Adaption to Big Databases

Hadoop is used which is mainly useful for large datasets, that can't be managed by single pc. It has distributed storage and any time any number of computers can be added or removed from this cluster. The files have replicas on different nodes thus making the system very reliable & resilient to failure. It solves the problem by divide and conquer approach and achieve parallel processing. Some of the algorithms have been adapted to big databases for parallel processing on multiple nodes using MapReduce Hadoop scripts in R environment.

MOOC "Managing Big Data with R and Hadoop" :
PRACE FutureLearn MOOC is a course on how to manage large amounts of data using Hadoop MapReduce in R environment. The data has been made available in HDFS. The script developed is how to do data formatting, aggregation and calculating mean and standard deviation. Then plotting the results.

Conclusions and Future Work

In this project, we use the electricity consumption data received from the Slovene company. We collect Weather and Holidays information and find their influence on electricity consumption. We perform clustering to find groups of days with similar consumption pattern. Using this clustering information, we construct a prediction model. We also adapt the data to NoSQl and RHadoop MapReduce framework for large datasets & parallel processing.
As a future direction, analysis can be extended to larger amounts of data which will improve the predictions many folds. Also, Advanced Clustering Algorithm (ACA) can be explored to reduce the cluster variances.

References

[1] Fazil Kayteza, M. Cengiz Taplamacioglua, Ertugrul Camb, Firat Hardalac (2014). Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines.

[2] Usman Ali, Concettina Buccella, Carlo Cecati (2016). Households electricity consumption analysis with data mining techniques.

PRACE SoHPCProject Title
Industrial Big Data analysis with RHadoop

PRACE SoHPCSite
University Of Ljubljana, LECAD Laboratory, Slovenia

PRACE SoHPCAuthors
Khyati Sethia, The University of Salford, United Kingdom


Khyati Sethia

PRACE SoHPCMentor
Janez Povh, University of Ljubljana, Faculty of Mechanical Engineering, Ljubljana, Slovenia

PRACE SoHPCContact
Khyati Sethia, University Of Salford
Phone: +44 7739897649
E-mail: khyati.sethia@gmail.com

PRACE SoHPCSoftware applied
R, RHadoop, MongoDB

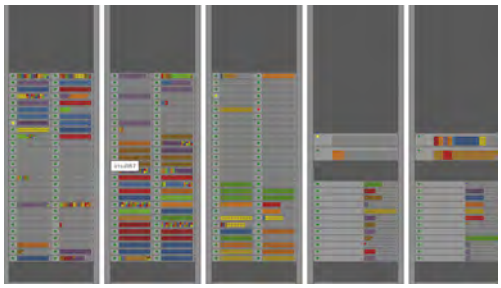
PRACE SoHPCMore Information
R, RHadoop, Future Learn, MongoDB

PRACE SoHPCAcknowledgement
The author would like to thank PRACE Summer of HPC for this opportunity. Also, author is grateful to Dr. Janez Povh and Dr. Leon Kos for their mentoring during this project.

PRACE SoHPCProject ID
1923

Energy reporting in Slurm Jobs

Matteo Stringher



HPC and Supercomputing centres are very intensive energy consumers. Energy reporting in such centres is a key feature of the system, especially for operations staff. During this project we have worked towards the implementation of a plugin to enhance the Slurm energy reporting capabilities.

Energy consumption is one of the main concerns in the HPC world. As order of magnitude, the power demand in the largest HPC centres is about 5 to 10 megawatts. In the race to Exascale-level systems, the HPC facilities have set a cap on the total power available, to ensure both efficient and economically-viable systems.

The power used for cooling the system can be roughly equal to the power consumed to keep up the nodes while under user workload. It is clear that measuring the consumption for each component of the system allows to better understand and optimise the work performed on the system. In the past years, more and more attention has been paid not only on performance parameters, but also on efficiency, such as a system's Gigaflops per watt metric, which is used for the Green500 ranking of supercomputers.

Tool set and project description

Slurm

Slurm is one of the main job scheduler and resource management system in HPC centres. It is adopted by most of the computer systems listed in the TOP500.

Energy consumption can be retrieved in two different ways: external sensors and internal monitoring. The project focused on the second methodology. Slurm, natively, provides support for IPMI and RAPL. The former is focused on baseboard management and consumption retrieval, while the latter operates at a more fine-grained level for Intel chips.

Slurm plugin development

Slurm allows external interaction with its functions in two different ways: the Slurm plugin API or SPANK. As regards the first one, Slurm provides a different interface for each type of task the plugin is meant for (e.g. accounting storage, energy).

SPANK which stands for Slurm Plug-in Architecture for Node and job (K)control, provides an easier approach, where there is no need to access the source code. In the project the latter has been chosen, since our requirements were satisfied by the SPANK capabilities (e.g. possibility to spawn a metrics collection process for the duration of a user's task).

Moreover, the installation is rather straightforward: once compiled the code into a shared library, the plugin can be easily mounted by adding a line to an internal configuration file - *plugstack.conf*, which controls where plugins are loaded from and how Slurm should deal with their exit states. It is suggested to compile it every time the Slurm installation on the operational system is updated. The configuration must be identical on each node where the plugin will be run.

NVML

In the past decade GPUs have been increasingly requested by scientific users, in fact, they allow to speed up a variety of codes, from molecular dynamics applications to training of deep neural networks. Deep learning training phases can be costly, especially, when an hyper-parameter analysis is needed to fine-tune the model. GPU spot power consumption can be retrieved for NVIDIA GPUs through IPMI or NVML (provided by NVIDIA). According to the NVIDIA documentation, it is possible to retrieve the power usage for the GPU and its associated circuitry.

To our knowledge, Slurm does not provide an integration for GPU power consumption recording, so we decided to build and experiment a plugin able to collect data and summarise the results to the user, such as the total consumption of his tasks that use GPUs.

Hierarchical Data Format

HDF5 is a popular file format in the

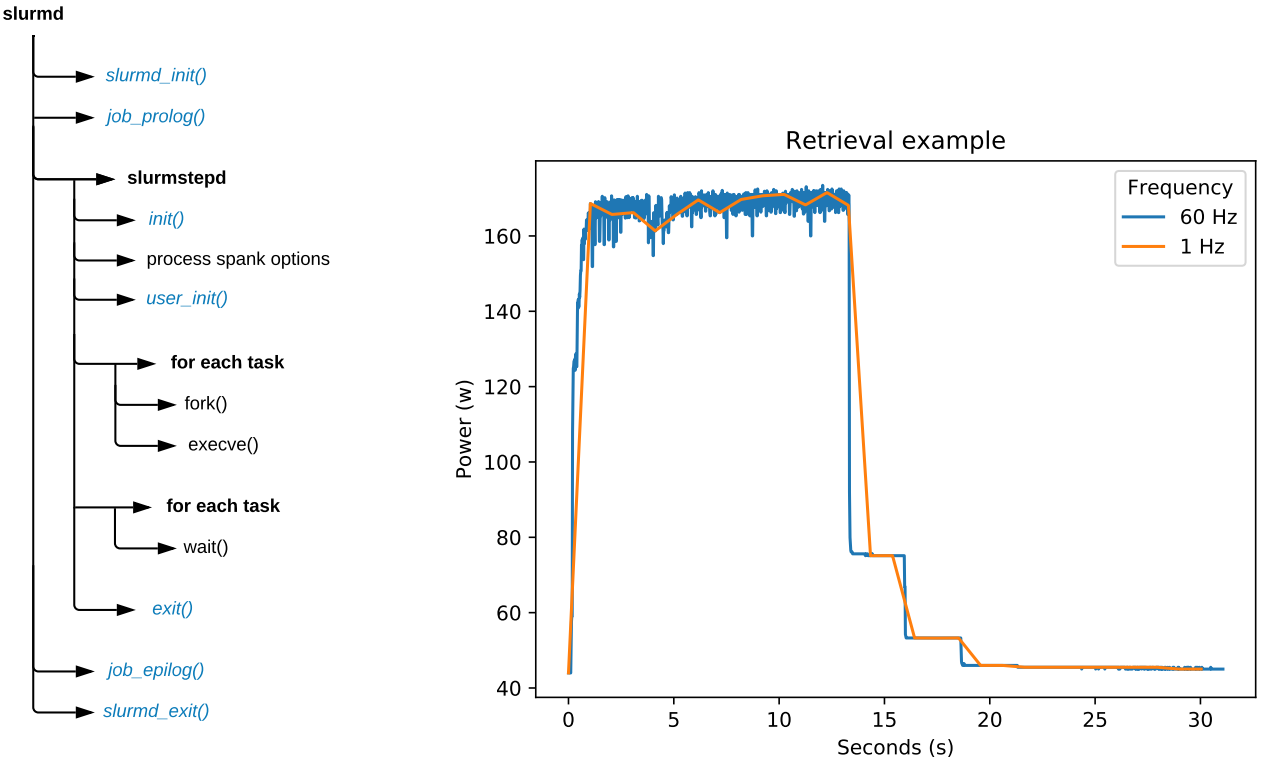


Figure 1: Left image: Simplified workflow of the SPANK functions that can be called inside the *slurmd* and *slurmstep* daemons. Right image: Demo run in the second test phase (before the installation of the plugin) on the *Iris* cluster to compare the use of different sampling frequencies.

scientific fields to store heterogeneous and large amounts of data in a single file. Moreover, it integrates with MPI for writing/reading data in parallel up to large scales.

Project development

Virtualized system and Slurm architecture

To develop the plugin a virtualized system has been adopted in order to work in a safe testing environment, without relying on a phsyical cluster. We have leveraged the Vagrant-VirtualBox combination in order to to deploy a virtualized HPC infrastructure. This improves debugging and development time. More information about the setup and micro-cluster composition is available in the repository documentation.[1]

The Slurm infrastructure is mainly composed by 3 entities: management, front-end and computation nodes. When launching a job, Slurm allocates the number of nodes requested by the user. One node with relative ID equal to 0 (a job's 'head node') manages task launches. Each computation node runs its own copy of the slurmd daemon, which is in charge of starting a slurmstepd daemon for each of a job's steps. Our plugin interacts with the slurmd daemons thanks to the SPANK library.

Slurm inbuilt energy accounting

Data collected through SLURM's IPMI/RAPL mechanisms is retrieved during the computation and the value of the final consumption is stored on the slurm database. This is actually a tunable option that must be set in the proper configuration file.

The *slurmdb* library offers different functions to query the database. Our plugin retrieves the value from the database at the end of the job and informs the user. The Slurm documentation highlights that the value must be considered only if the node has been reserved in exclusive mode, such that a job's consumption is accurately tracked.

SPANK integration

SPANK easily allows to modify the job behaviour. Our plugin can be invoked from the *sbatch* command, when launching the job script. Along with the *–energy-reporting* flag - which activates energy measurements - the user can specify the sampling frequency (expressed in hertz), otherwise a default value of 20 measurements per second will be used.

SPANK provides an interface, which must be used to let the *slurmd* and *slurmstepd* daemons to invoke the plugin. SPANK plugins are loaded in up to five separate contexts: *local*, *remote*, *allocator*, *slurmd*, *job_script*. *Remote* was the mainly used one, since most of operations are executed inside the step level. The image 1 at the left shows the different calls that can be used with the SPANK interface.

At each step of the job script, a process is forked on every node, which is in charge of storing on the HDF5 format file the timestamp and the measured power consumption, i.e. one timestamp and one measurement for each GPU mounted on the mainboard. The sampling frequency must be equal or greater to 1. All the samples are stored on a buffer which is flushed to the HDF file every second.

When the user task ends, the forked monitoring process is stopped. The HDF file will then contain several datasets, one for each step and for each GPU.

In each dataset, two sequences of n samples can be used to calculate the consumption over the time. Given

[1] puppet-slurm: github.com/ULHPC/puppet-slurm

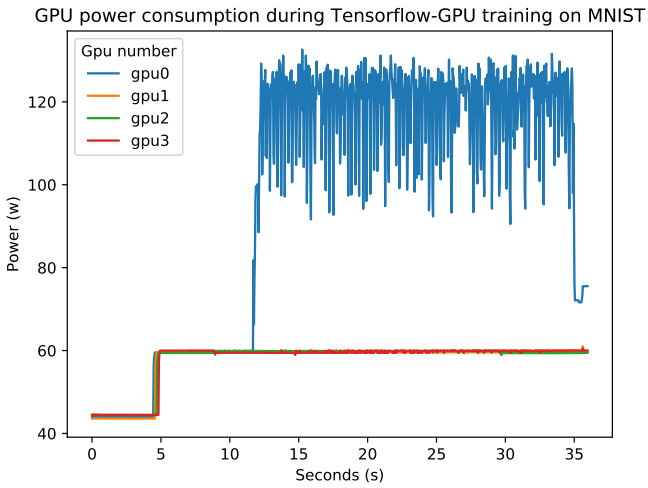


Figure 2: MNIST convolutional NN training test.

$P_{1,\dots,n}$ the samples collected at each timestamp $t_{1,\dots,n}$, the total consumption can be approximated by the equation:

$$E_{tot} = \sum_{i=2}^{n} (t_i - t_{i-1}) \frac{P_i + P_{i-1}}{2}$$

Using the equation above the consumption of each GPU of each node reserved will be computed, and the final value is shown to the user expressed in Joules.

Results & further work

The implemented plugin can seamlessly interact with the Slurm execution. Figure 1 at the right shows the power consumption for the same run with different sampling frequencies. It can be seen that the choice of the sampling rate has a notable effect, the blue curve has a lot of spikes, which can not be highlighted with a lower frequency. It must be underlined, that the precision of the measurement provided by the NVML library is subject to error.

The plugin has been developed and tested through three stages.

- In the first one we have used the virtualized cluster to ensure to run a safe code on the physical one. This development part took most of the project time.
- In the second one a prototype of the code has been run on the *Iris* cluster without affecting the Slurm configuration. The results are shown in Figure 1 at the right. The core data retrieval functions at this stage were able to collect information from all the GPUs on the mainboard.
- In the last phase the plugin was fully assembled and tested by installing it on the cluster and executed on compute nodes featuring dual *Skylake* CPUs and *quad Volta V100 GPUs*.

Figure 2 shows a Slurm job executing a Singularity container with GPU-enabled Tensorflow 1.12 and Python3. The same code has been tested on a longer run achieving similar results. In detail the job steps are:

- 0-5s: loading software environment (lmod), Singularity container initialization
- 5-12s: TensorFlow-GPU initialization, data load disk to memory
- 12-36s: training for 10 epochs
- job spindown

Future work could study and understand the overhead of the retrieval process. This runs mostly on the CPU, so we expect the overhead to be low for GPU-intensive processes.

Conclusions

We have prototyped a new Slurm plugin for energy reporting of a user's tasks, which can interact with NVIDIA GPUs through the NVML interface, allowing for GPU energy consumpton retrieval - a novel development to our knowledge. The code is available as open-source.[2]

The energy consumption data is stored for further analysis if needed. The HDF5 file format supports large datasets and compression, and is used by our plugin to store timestamped energy measurements. The plugin manages parallel jobs and all the contained steps. The user is informed about the energy usage for each job step and precise identification of hot spots is possible (together with other information from Slurm accounting).

PRACE SoHPCProject Title
Energy Reporting in Slurm Jobs

PRACE SoHPCSite
University of Luxembourg, Luxembourg

PRACE SoHPCAuthors
Matteo Stringher

PRACE SoHPCMentor
Valentin Plugaru, Unilu, LU


Matteo Stringher

PRACE SoHPCContact
Matteo, Stringher, University of Luxembourg
E-mail: stringher.matteo@gmail.com

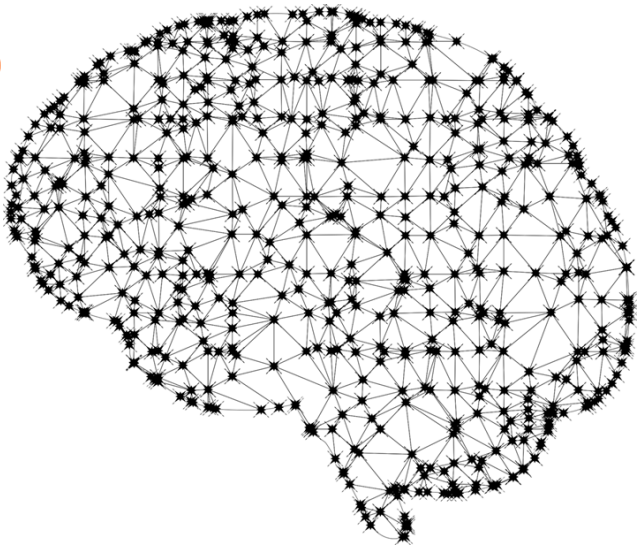
PRACE SoHPCSoftware applied
Slurm, Vagrant, Puppet, Virtualbox, C

PRACE SoHPCAcknowledgement
I gratefully acknowledge the PCOG group for the project and the support given before and throughout the whole summer.

PRACE SoHPCProject ID
1924

[2] gitlab.uni.lu/sohpc/public/slurm-spank-energy-reporting

Benchmarking Deep Learning



Sean Mahon

This project deals with different ways of evaluating the performance of distributed training of Deep Learning models and comparing the efficiency of multiple widely used Frameworks. In particular, experiments were run to determine the scalability of training Resnet models on the CIFAR10 Dataset using both CPU and GPUs.

Deep learning is becoming an extremely prominent method of solving problems both in academia and in industry. However, training deep neural networks, which often contain hundreds of thousands, if not millions of parameters, can be extremely computationally expensive. Therefore, it is natural to attempt to distribute this training over multiple processors where possible. There are multiple ways of approaching this task with the simplest and most common being data parallelism.[1]

Put simply, data parallelism involves splitting up the sample of the data, or batch, used in each training step. between all available processors. The resulting changes in the trainable parameters of the model are then averaged across all processors before the process is repeated. However, due to the high number of parameters to be communicated between processors, it is rarely optimal to keep the same hyperparameters when adding more devices. Normally, this is achieved by fixing the batch size per device and adjusting other parameters to compensate.

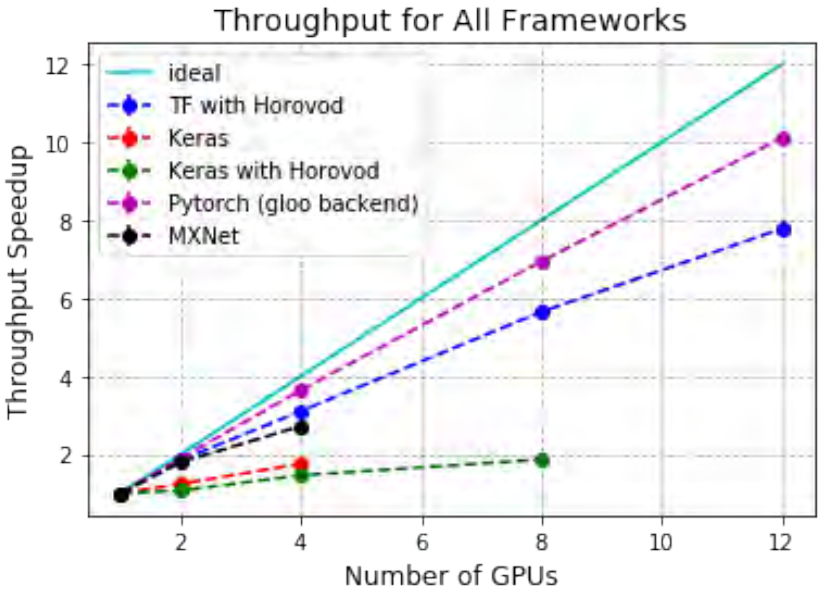
This creates a problem when considering how to evaluate how well a model is performing. Calculating the speedup in throughput, or the number of data points processed per second, is a common approach taken by chip manufacturers but may not be helpful if the model does not train as effectively on too many devices. For this reason, many existing ML benchmarking efforts, such as MLPerf, place considerable emphasis on time-to-accuracy results. This involves measuring the walltime required for the model to reach a certain level of accuracy.

Further complications arise from the fact that a number of different libraries and frameworks exist for deep learning, many with very similar functionality. It is important to note that, while these frameworks share many of the same functions, the implementation can vary somewhat between frameworks. While there have been some efforts to create benchmarking tools which work with multiple frameworks, such as Deep500,[2] there does not seem to be any individual benchmarking platform which provides comprehensive support for all of the most common frameworks in use.

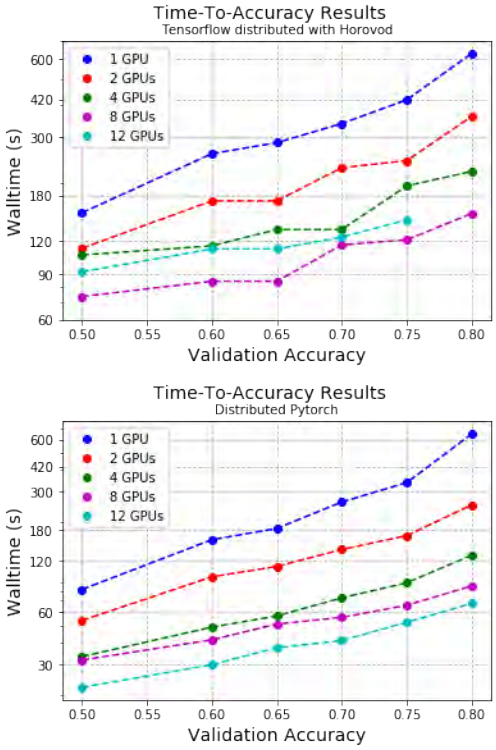
Structure of the Code

In order to compare the performance of different frameworks on the University of Luxembourg's Iris Cluster, software was written train models in a variety of frameworks while collecting data relevant to benchmarks such as those described previously between epochs. The current version allows the user import a model and dataset and supply a config file with details of the training before it automatically distributes everything it across the hardware specified, trains it for some given number of epochs, and collects common benchmarking metrics at the end of the run. In addition, there is also a script to sort parse the config file and produce a suitable SLURM batch script to ensure that resources allocated match what is specified. The frameworks supported are Tensorflow (distributed with Horovod), Keras (native `multi_gpu_model` and Horovod), MXNet, and PyTorch (`DistributedDataParallel` Models with Gloo backend)

As mentioned in the last section, there are multiple variables which affect whether the training process can be parallelised efficiently. Some research and experimentation was required to find sensible default settings for the scaling of hyperparameters. However, in general, it was found that current best practice as described in the literature[3] (and well summarised in this article) was effective.



(a) Speedup in Throughput for all Frameworks Considered



(b) Time-to-Accuracy curves for Tensorflow and Torch

Figure 1: Results from GPU Experiments

Sample Problem: Resnet and CIFAR10

While the code written for this project could in principle be used for a variety of different models and tasks, it may be insightful to consider a well known example for experiments. The dataset used was CIFAR10,[4] a standard collection of 32×32 pixel images featuring 10 categories. The dataset contains 50,000 images to train a model on in order to correctly identify the category of object in a further 10,000 unseen test images.

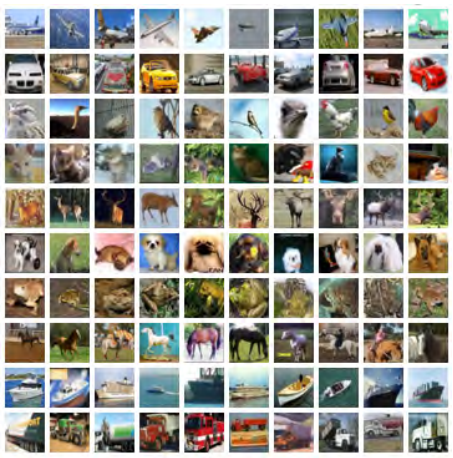


Figure 2: Sample Images from CIFAR10

The model used to classify these images was a version of Resnet,[5] a well known example of a Convolutional Neural network for image classification problems. In particular, a 44 layer network, with over 600,000 trainable parameters was used. While training such a model may sound like a substantial task, this would be considered to be a medium sized problem in deep learning. While it would be inadvisable to attempt to train this model on a normal laptop, it would not necessarily take several hours to get meaningful results when using more sophisticated hardware.

GPU Results

This model was trained for 40 epochs with a local batch size of 256 using the frameworks mentioned for using varying numbers of GPUs. It should be noted that the same data pipeline was used to load and perform simple augmentations on each batch to ensure inconsistencies in data used do not affect training results. The mean speedup in throughput is shown in figure 1a. The main trends visible for each framework were:

- The PyTorch results (found using the `DistributedDataParallel` functionality and gloo backend) seem to scale very well and only begin to substantially deviate from ideal predictions once more than one node (4 GPUs) is used.
- Tensorflow distributed using Horovod also scales well though inefficiencies are visible at a lower number of GPUs than for pytorch. It should be noted that, once serial code has been implemented, distributing training with Horovod requires very little extra code compared to other frameworks.
- Throughput for MXNet increased at a slightly lower rate than Tensorflow and PyTorch with moderate inefficiencies visible when using just 4 GPUs. Distributed training was also less user-friendly to setup than for other frameworks, meaning support for multiple nodes has yet to be implemented.
- The experiments for Keras did not appear to scale well. This is not surprising as it is the most high level of the frameworks considered and Horovod was originally designed to work with Tensorflow. There is also the issue that the native `multi_gpu_model` only executes training steps in parallel rather than loading data, potentially creating extra bottlenecks

As mentioned previously, while examining the throughput is a very concise, scientific, way to determine how fast a program is running, it does not give much indication as to whether the efficiency of the training has been adversely effected by adding more devices. For this reason, plots of the walltime required for the model to reach various test accuracies for the two frameworks with the highest throughput are shown in 1b. From examining these graphs, it appears that once 4 or more GPUs are used, the benefits of adding more seem somewhat limited. Note that Tensorflow seemed to actively slow down and did not reach 80% accuracy in its 40-epoch run once 12 GPUs were used. This is likely due to the fact that the changes to the training hyperparameters, such as increasing the global batch size, eventually start to prevent the model from training effectively.

It also appears that the curves for Horovod are not quite as smooth as those for PyTorch. A possible explanation of this is that Horovod does not synchronise non-trainable weights in the model's batch normalisation layers. While these weights should converge to common values eventually, it is plausible that discrepancies between processors would add noise to results from the early stages of the training.

CPU Results

In the case of the two most promising frameworks, similar experiments were run using CPUs instead of GPUs for comparison. To avoid memory errors, the batch size had to be reduced to 16 per CPU. As may be expected, these took much longer to run. However, as can be seen in figure 3, the throughput seems to scale much at least as efficiently as the GPU version. PyTorch seemed to stay within a reasonable margin of error to ideal scaling, up to the 4 nodes (112 CPUs considered). A likely reason for this is that, as using a single CPU for training would be substantially slower than one GPU, communication is taking up a smaller fraction of the total training time in these experiments.

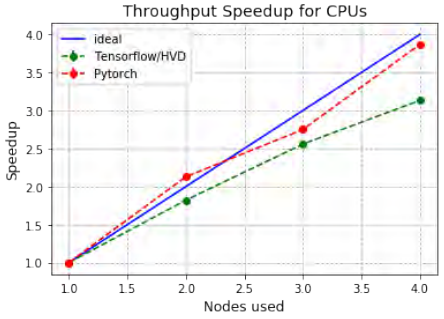


Figure 3: Throughput Speedup from CPU Experiments

However, as before, the efficiency of training seemed to be impacted significantly by the high global batch size which accompanies the use of many processors. As seen in figure 4, improvement in time-to-accuracy results is negligible when more than 56 CPUs are used. Note also that y axis scale suggests that the performance in this limiting case is similar to what would be expected from using two GPUs.

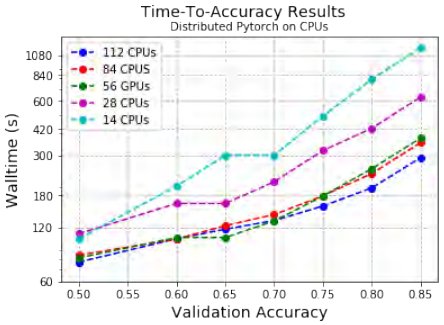


Figure 4: Throughput Speedup from CPU Experiments

Conclusions

While some aspects of the results found are likely to be specific to this particular neural network, it appears that, for problems of this type. The `DistributedDataParallel` features in the PyTorch library seem to be the most efficient of the frameworks considered for distributing the training process across many devices. Tensorflow with Horovod also scales well and was easy to use but it appears that some of the shortcuts taken to improve throughput had a negative effect on test accuracy. The other frameworks tried generally did not scale as well as these two. It should be noted that, for both PyTorch and Tensorflow, it was evident that some of the common changes to hyperparameters, such as increasing the batch size, caused the time-to-accuracy results to stop improving once too many processors were used, at least in this case.

Similar results were found when using CPUs and GPUs. In the case of the former, little improvement was seen in the time-to-accuracy results once more than 56 CPUs were used. However it was noted that, despite being lower than the GPU case in general, the speedup in throughput when more CPUs were used was very close to ideal, likely due to the proportion of time spent on communication being lower. From this fact, it may be reasonable to claim that, if the problem is sufficiently large and is not particularly sensitive to changes in batch size and other hyperparameters, it would be possible to achieve near ideal scaling using the correct framework.

References

[1] Géron, Aurélien (2017), Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems, O'Reilly Media

[2] Ben-Nun, Tal and Besta, Maciej and Huber, Simon and Nikolaos Ziogas, Alexandros and Peter, Daniel and Hoefler, Torsten (2019), A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning

[3] Goyal, Priya and Dollár, Piotr and Girshick, Ross and Noordhuis, Pieter and Wesolowski, Lukasz and Kyrola, Aapo and Tulloch, Andrew and Jia, Yangqing and He, Kaiming (2017), Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

[4] Krizhevsky, Alex(2009) Learning Multiple Layers of Features from Tiny Images

[5] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun (2015) Deep Residual Learning for Image Recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (770-778)

PRACE SoHPCProject Title

Performance analysis of Distributed and Scalable Deep Learning

PRACE SoHPCSite

University of Luxembourg, Luxembourg

PRACE SoHPCAuthors

Sean Mahon, Trinity College Dublin, Ireland

PRACE SoHPCMentor

Sebastien Varette, University of Luxembourg; Luxembourg; Valentin Plugaru, University of Luxembourg, Luxembourg

Sean Mahon

PRACE SoHPCAcknowledgement

I would like to thank my project mentors Sebastien Varette and Valentin Plugaru, as well as the other researchers and students in the UniLu HPC group for their continued help and support during my time in Luxembourg.

PRACE SoHPCProject ID

1925

www.summerofhpc.prace-ri.eu