PRACE

Partnership for Advanced Computing in Europe

2022

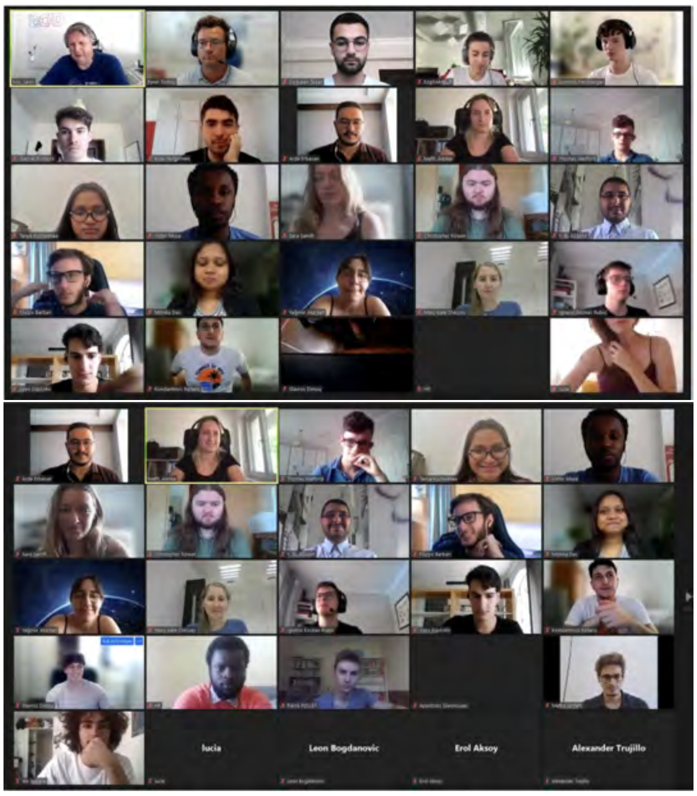
summerofhpc.prace-ri.eu

A long hot summer is time for a break, right? Not necessarily!
PRACE Summer of HPC 2022 reports by participants are here.

Hybrid 2022!

Leon Kos

Summer of HPC 2022 started hybrid (online and onsite) with 30 participants and their mentors at 11 PRACE HPC sites working on 21 projects.



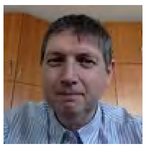
Summer of HPC is a PRACE programme that offers summer placements at HPC centres across Europe to late-stage undergraduate and master's students. Total of 30 top applicants from across Europe were selected to participate in pairs or onsite on 21 projects supported and mentored from 11 PRACE hosting sites.

Participants spent two months working on projects related to PRACE technical or industrial work and produce a report and video of their results. A kick-off online training week (see photo) was organised by University of Ljubljana.

At the end of the summer videos were created and are available on Youtube as PRACE Summer of HPC 2022 presentations playlist. Together with the following articles interesting code and results are available. Dozens of blog posts were created as well.

Contents

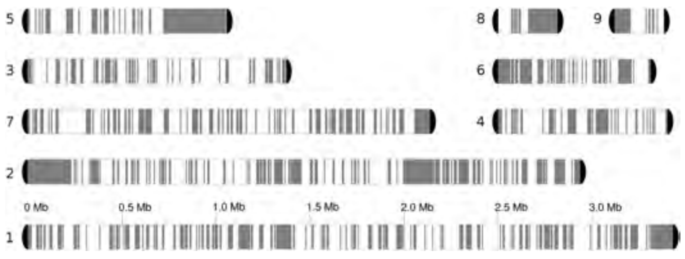
PRACE SoHPC2022 Coordinator
Leon Kos, University of Ljubljana
Phone: +386 4771 436 E-mail: leon.kos@lecad.fs.uni-lj.si
PRACE SoHPCMore Information
http://summerofhpc.prace-ri.eu


Leon Kos

Leveraging HPC to test quality and scalability of a genetic analysis tool

Genome data, its loss and HPC

Victor Njenga Muya



Heterozygous regions are in white, while homozygous blocks of at least 5 kb are in grey.[4]

JLOH is a tool that analyzes DNA to identify regions that have lost some information through loss of heterozygosity. This project aims to optimize the performance of JLOH and make it fast, efficient and scalable on an HPC cluster.

When two divergent organisms - belonging to different species, subspecies, etc - interbreed, their offspring are said to be hybrid. Hybrids possess copies of genetic material from both parents in their genomes, and are therefore heterozygous,[2] see figure 1. Common examples of hybrids today are wheat, quinoa, cabbage, and tobacco among others which are prominent in the agricultural sector. Fish, fungi, and some higher eukaryotes for example yeasts, and donkeys are hybrids too.

Heterozygosity may come as an advantage to these organisms since they have alternative alleles for some genes and either of these alleles could be picked during their evolution where natural selection is stringent. However, this is not always the case. The fact that they carry different alleles in their subgenomes also creates imbalances in the organism's cell dynamics as the two subgenomes cannot easily pair during cell division. Consequently, throughout its lifetime, the hybrid genome undergoes rapid changes called rearrangements that may result in the damage, loss, or doubling of DNA in some regions of the genome. The process through which one of the alleles in a heterozygous site of a genome is lost and replaced with a copy of the other, or simply just deleted is known as **loss of heterozygosity**.

Usually, most of these changes are deleterious and could decrease the hybrid's overall fitness. For instance, loss of heterozygosity in human tumor suppressor genes increases the risk of cancer development. On the other hand, LOH events could also be beneficial where the hybrid grows faster and adapts to its environment better than its parents. For example, hybrid yeast species used in alcohol fermentation tend to produce more ethanol and ferment alcohol faster than their parents under the same conditions. Interest in LOH has grown considerably both in academia and industry since it is a marker for the evolution and adaptation of hybrid species. However, little to no software exists for the extraction of LOH data from DNA sequencing data. To fill this gap, the Comparative Genomics Lab at the Barcelona Supercomputing Center has developed JLOH, a genetic analysis toolkit, that extracts blocks from LOH from sequencing data, which is the subject of this project. Starting with single nucleotide polymorphism (SNP) data, JLOH identifies regions in a genome where stretches of the differences between two genomes have been lost through LOH. The core of the JLOH algorithm is written in Python, but the tool also has a Nextflow pipeline that enables it to scale on a cluster.

JLOH Input

A DNA strand is a double-stranded sequence of the molecules: Adenine (A), Thymine (T), Guanine (G), and Cytosine(C), which are paired in a double helix as AT and GC and are collectively called nucleotides. Just like the order of letters in a word determines its meaning, the order of these bases encodes instructions for the cell machinery to build proteins, and is what we call a "gene". Genes make up a chromosome, and several chromosomes constitute a genome (eukaryotes). When analyzing a genome, its exact genome sequence is established through DNA sequencing. In

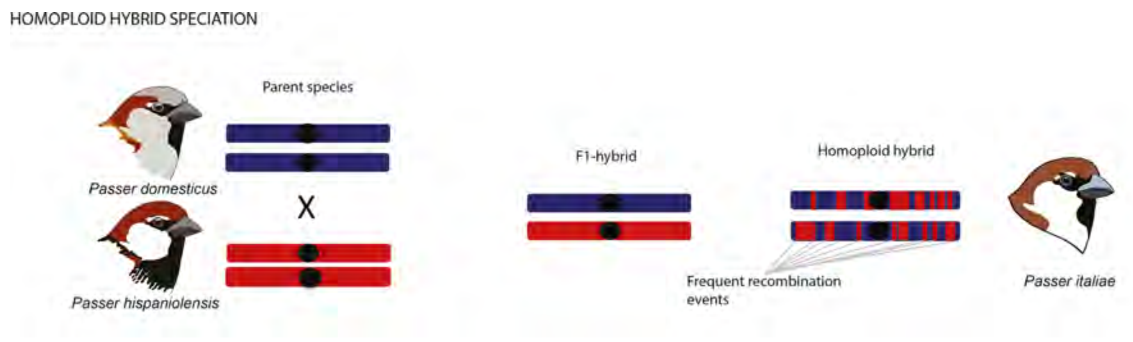
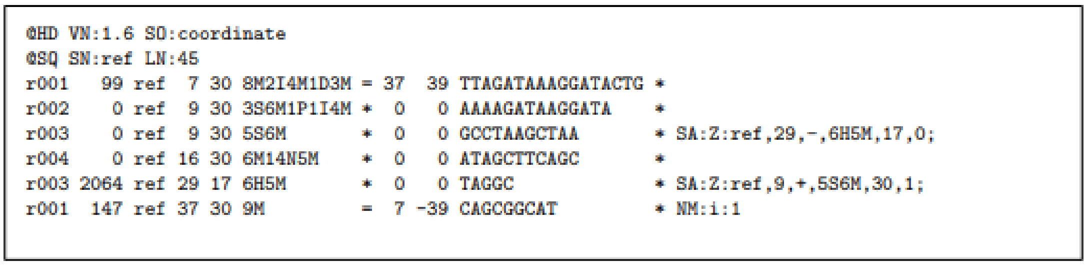


Figure 1: Evolution of Hybrid species from parents of different species[2]



```
@HD VN:1.6 SO:coordinate
@SQ SN:ref LN:45
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC             * SA:Z:ref,9,+,5S6M,30,1;
r001  147 ref 37 30 9M         =  7 -39 CAGCGGCAT         * NM:i:1
```

Figure 2: The SAM format[3]

this project, we are working with paired-end Illumina sequencing reads of the hybrid yeast of the species ***Saccharomyces cerevisiae*** and ***Saccharomyces uvarum***. JLOH uses the following genomic file formats as inputs

- **FASTQ** - The raw sequencing reads come as a pair of FASTQ files. FASTQ is a text format containing the information related to each sequencing read organized in four lines: 1) an identifier, 2) the actual sequence, 3) a separator "+" sign, and 4) an ASCII encoded quality score for each sequenced nucleotide.
- **FASTA** - A text-based format that, just like FASTQ, stores reference sequences using the letters A, T, C, and G to represent the individual bases. Unlike FASTQ, they don't come with any quality score, only an identifier and the sequence itself.
- **Binary Alignment Map (BAM)** - This is the compressed binary version of the Sequence Alignment Map (SAM),[3] that is lighter to access and process despite not being human readable. The text-based SAM format, figure 2, stores the locations of the individual sequencing reads within the genome and tends to be enormous.
- **Variant Call Format (VCF)** - This is a tab-delimited text format that holds the variants and where they occur within the genome. .

In hybrid mode, the JLOH algorithm needs three inputs:

1. Two FASTA files - storing the reference sequences of the parent species of the hybrid.
2. Two BAM files mapping the records of the hybrid reads onto the parental genomes
3. Two VCF files containing the SNPs representing the variants found between the hybrid and the two parental genomes

The raw FASTQ files can be passed to the Nexflow pipeline and JLOH would work and give the desired output. Armed with these input files, JLOH outputs the blocks in the parental genomes where some alleles have been lost or deleted through LOH as several output files.

The JLOH Algorithm

JLOH performs its analysis in several steps. It:

- Separates homozygous and heterozygous SNPs and writes them in two separate VCF files.
- Calculates the homozygous and heterozygous SNP densities in both parental genomes, and clusters the regions of high heterozygosity and homozygosity together.
- Identifies the heterozygous and homozygous regions of low SNP density, labeling them REF and ALT blocks. This is where LOH has occurred as these sites were heterozygous initially.
- The REF and ALT blocks are then compared to the heterozygous SNP clusters and the overlaps trimmed
- Creates coverage profiles for the REF and ALT blocks by analyzing these blocks for read coverage, and trimming any uncovered regions. Blocks whose coverage is above a set threshold are discarded. These profiles are used to determine each block's zygosity (whether hemi or homo). Homozygous regions display a uniform read coverage, while hemizygous blocks show a reduced coverage as compared to neighboring regions.[4]

JLOH then outputs a table of all the LOH blocks found, their zygosity (homo or hemi), their allele type (ALT or REF), mean coverage per position, and coverage ratios as the main output file. Together with the input BAM files, these output files can be viewed in a genomics viewer, figure 3.

Profiling for efficiency

My work in this project was to establish how efficient each of the functions

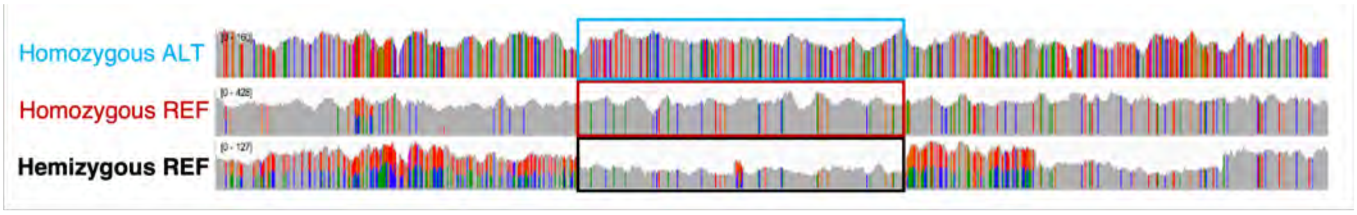


Figure 3: JLOH output viewed in a Genomics Viewer[1]

within JLOH is. Using different Python tools, I conducted profiling tests where I tested how JLOH performs by measuring how much time and memory each of the functions within the program takes to run relative to the total run time and memory consumption. With this information, we were able to identify the functions that consume the most time and we tried to optimize them where it was possible.

Scalability: Computational Time

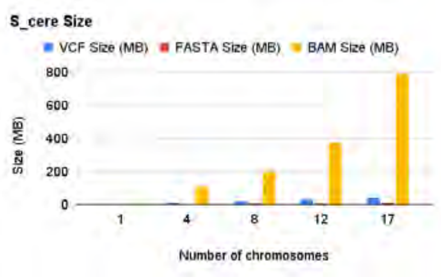


Figure 4: Size of the data sets in MB and number of chromosomes in each data set

JLOH makes extensive use of parallelization within its functions to perform its analysis. Using Python's multiprocessing module, JLOH leverages the power of multiple CPU cores to run its functions in parallel using multiple processes and considerably cut down the run time. To test whether JLOH computational time is scalable, I ran it as job scripts using the data subsets of incremental size, see figure 4, with a fixed amount of memory, and CPU cores, and observed how much time was spent to complete the analysis each time.

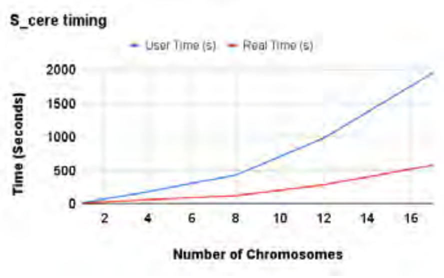


Figure 5: User and real time recorded from running JLOH with increasing data

In figure 5, real time (red) is the time it takes from when the job script starts running until the analysis has run and we have output as if timed using a stopwatch. User time (blue) is the cumulative time taken by the CPUs to run the program. This is the amount of time it would take had JLOH been running serially (on a single CPU), clearly showing that JLOH is running in parallel. Moreover, the computational time increases linearly as the size of the input data increases. JLOH's computational time is scalable!

Scalability: CPU Cores

Using a fixed amount of memory and a single data set, I tested how scalable JLOH using by varying the number of CPU cores. The user time (red) remained constant but the real-time (blue) decreased as the number of CPU cores increased eventually flattening out at 4 cores, for this data set ,see figure 6.

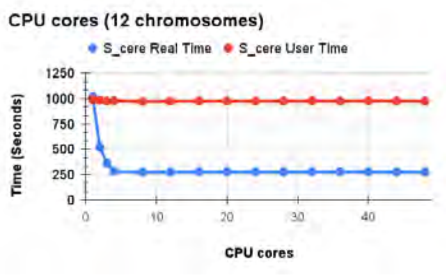


Figure 6: The time taken by JLOH to analyse the 12 chromosome data set with a varied number of CPU cores

When tested with 10 CPUs, a fixed data set, and a varying amount of memory per CPU, the user and real times remain steady indicating that increasing the amount of memory did not decrease the run time of the program. JLOH is parallelized to take advantage of CPUs and consumes little memory to run its analysis.

Conclusion

Overall, JLOH performed well across the different tests that we conducted, proving to be scalable based on computational time, CPU cores, and light on memory consumption. The toolkit's parallelization strategy proved effective hence the difference between the real and user times.

References

[1] GitHub - gabaldonlab/jloh: A tool to extract loh blocks from snps. https://github.com/Gabaldonlab/jloh JLOH GitHub Repository: https://github.com/Gabaldonlab/jloh

[2] Runemark, A., Vallejo-Marin, M., & amp; Meier, J. I. (n.d.). Eukaryote hybrid genomes. PLOS Genetics.

[3] Sequence alignment/map format specification - https://samtools.github.io/hts-specs/SAMv1.pdf

[4] Pryszcz et al. (2015) The Genomic Aftermath of Hybridization in the Opportunistic Pathogen Candida metapsilosis. PLoS Genet 11(10): e1005626.

PRACE SoHPCProject Title
Leveraging HPC to test quality and scalability of a genetic analysis tool

PRACE SoHPCSite
Barcelona Supercomputing Center, Spain

Victor Njenga Muya

PRACE SoHPCAuthors
Victor Njenga Muya, Osmangazi University, Turkey

PRACE SoHPCMentor
Toni Gabaldón, BSC, Spain
Matteo Schiavinato, BSC, Spain

PRACE SoHPCContact
Victor Njenga Muya, Osmangazi University
Phone: +90 552 269 2369
E-mail: victornjenga98@gmail.com

PRACE SoHPCSoftware applied
Python, Nextflow, Github

PRACE SoHPCMore Information
JLOH Github Repository

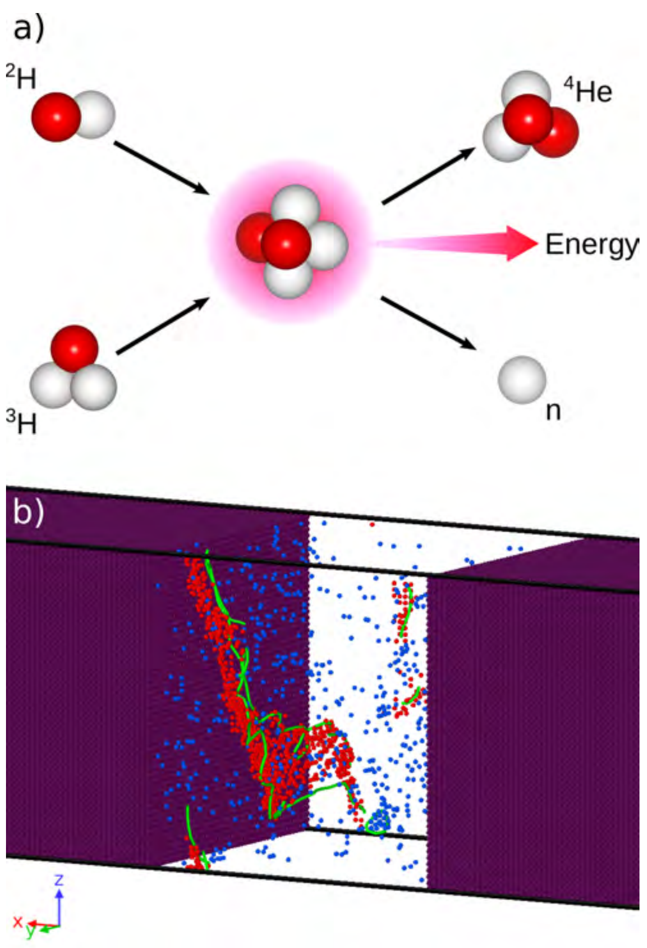
PRACE SoHPCAcknowledgement
I am grateful to my project mentors Toni Gabaldón, and Matteo Schiavinato who guided me through the project, and everyone at the Comparative Genomics Lab and BSC for hosting me. I am also grateful to PRACE for making SoHPC possible and allowing me to work on this project.

PRACE SoHPCProject ID
2201

Tungsten Simulations for Nuclear Fusion

Arda Erbasan, Dominik Freinberger

Fusion is considered to be one of the most promising energy sources of the future. It is known that the plasma inside the fusion reactor must reach a temperature of millions of degrees. Moreover, the radiation levels around the plasma are incredibly high. What kind of reactor should be designed to obtain energy from such an environment so that it can withstand these harsh conditions?



Cover Figure: a) An Illustration of the Fusion Reaction, b) Defected simulation box: red, blue, and purple balls represent interstitials, vacancies, and undamaged Tungsten atoms, respectively.

Nuclear Fusion is a promising candidate for a safe, sustainable, and carbon-free future energy source. However, what our sun and stars have been doing for billions of years is far more challenging to realize here on Earth. Under terrestrial conditions, the most promising process for generating usable net energy is the fusion of deuterium-tritium, two isotopes of hydrogen. Figure a) on the cover illustrates this reaction, with a helium nucleus and a neutron as the fusion output and a lot of energy released, most of which comes as kinetic energy of the two reaction products. To start and sustain the fusion process, the fusion fuel, deuterium, and tritium, must be heated to several million Kelvin, causing the electrons to separate from the atomic nuclei and the fuel to enter the plasma state. Although the plasma in a fusion reactor is magnetically confined in a vacuum and suspended, there are high demands on so-called plasma-facing components (PFC). These reactor materials are directly exposed to high radiation, immense heat load, and neutron bombardment. Needless to say, materials intended to withstand these harsh conditions must exhibit unique properties, such as a high melting point, high thermal conductivity, and a long lifespan, for economic and safety reasons. Good thermal conductivity is essential to dissipate a high amount of heat quickly and effectively to a coolant. One such material possessing the desired properties and already used in fusion reactors is Tungsten, whose thermal conductivity after irradiation is the primary research subject in our SoHPC project.

Methods

Molecular Dynamics

This work uses Classical Molecular Dynamics, a computer simulation technique to simulate atoms' and molecules' movements and thermodynamic properties. In molecular dynamics, the interacting pieces are atoms, and they are modeled as point particles. The interaction of the atoms is described with interatomic potentials, which are functions of the atomic environment. The forces acting on each atom and the (potential) energy of the system can be derived from the potentials. Once the forces are known, the time evolution and hence the trajectory of the particles can be calculated based on Newton's Second Law, $\vec{F} = m\vec{a}$. From known quantities such as positions, forces, and momentum, physical properties can be deduced by computing temporal/spatial averages and other correlations. There are three main statistical thermodynamic ensembles we use in this study, and these are:

- Isothermal-Isobaric Ensemble (NPT): In this thermodynamical ensemble the number of particles (N), pressure (P), and temperature (T) are kept constant.
- Canonical Ensemble (NVT): The number of particles (N), volume (V), and temperature (T) are constant.
- Microcanonical Ensemble (NVE): In this step, the number of particles (N), volume (V), and total energy of the system (E) are held constant.

The software we are using to run the molecular dynamics simulations is called *Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS)* which is an open-source code.

Thermal Conductivity

The thermal conductivity of a material is a measure of its ability to conduct heat. For a given material, it can be computed with the aid of Fourier's law $\vec{q} = -\kappa\nabla T$ where $\vec{q}$ is the heat flux and ∇T describes a spatial temperature gradient. Thus, knowing $\vec{q}$ and ∇T enables us to calculate the thermal conductivity κ. One possible way of doing so is to define two distinct regions in the simulation box and add/subtract the same amount of energy to/from the two regions, respectively establishing a known heat flux $\vec{q}$ and a temperature gradient ∇T between the resulting hot and cold region.[1,2]

Creation Relaxation Algorithm

We use the Creation Relaxation Algorithm[3] (CRA) to simulate radiation damage inside the metal at a relatively low computational cost. This method first displaces a random atom in the crystal to a new random position. The displaced atom creates a vacancy in the initial position and a self-interstitial atom in the new position. These vacancy-interstitial pairs are also called Frenkel pairs. Then, energy and force minimizations are performed, and the atoms settle to their new energetically favorable positions. Finally, these steps are repeated until a desired level of damage is reached, as measured by the number of displaced atoms. In the course of thermal equilibration to reach the desired temperature under the different thermodynamic ensembles, the recombination of vacancies and interstitials produced by CRA takes place to some extent. This leads to a smaller number of final defects as compared to the created initial defects. An example of such defects is given in Figure b) on the cover page.

Simulation Setup

A simulation box of rectangular prism shape and dimensions of 158 x 15.8 x 15.8 nm consisting of 2.5 Million individual atoms is created, see Figure 1. The damage is introduced to the middle 10% of the simulation box with CRA. Then, the system's temperature is increased to the desired value while keeping the pressure constant. While this happens, the volume of the simulation box changes until it converges to a value for a given pressure and temperature. After obtaining the well-converged system, we put the system into the canonical ensemble. The main reason we apply this step after thermally equilibrating the system is to obtain a statistically probable state of the system under given conditions. Later, we construct an equidistant heat source (hot region) and heat sink (cold region) to the center of the simulation box. This is achieved by withdrawing kinetic energy from the heat sink and supplying the same amount of kinetic energy to the heat source. To obtain a steady state temperature gradient between "hot" and "cold" regions, we distribute the energy along the simulation box in a microcanonical ensemble. Before computing the thermal conductivity, the system was brought to a desired temperature using the thermodynamical ensembles described above, starting with NPT then NVT and finally NVE.

Results

Figures 2 and 3 show the temperature profile of a 300 K Tungsten system without defects versus with 1038 displaced atoms after recombination, respectively. The highlighted part in the center of the graph corresponds to the middle section of the simulation box. In the undamaged system, the temperature profile is smooth in the area of interest. In contrast, the temperature profile in the damaged system exhibits a sudden change in slope, indicating a change in thermal conductivity. For the perfect system, we computed a thermal conductivity of 18.59 W/mK, and for the damaged system, we obtained 10.70 W/mK. To better quantify this behavior, we analyzed the trend in thermal conductivity with respect to the overall damage of the system as measured by the final number of defects. As can be seen in Figure 4 (a), an increase in the number of defects comes with a decrease in thermal conductivity. We also observed some deviations from the main trend - possibly due to the statistical nature of the simulations. The numerical data can be found in the following Table, sorted by the final number of defects.

Table 1: Thermal conductivity at 300 K

Defects		
Initial	Final	κ(W/mK)
0	0	18.59
36590	405	12.33
32794	802	12.42
7022	933	11.17
8758	1038	10.70
9461	1137	10.92
29039	1142	10.86

Next, we advanced the investigation to 600 Kelvin. Although 600 K is not a high temperature for a metal whose melting point is above 3500 K, the procedure has to be slightly different compared to lower temperatures. As mentioned above, the thermal equilibration process includes constant pressure & temperature steps for which the volume changes

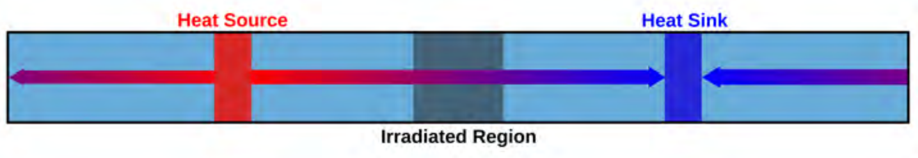


Figure 1: Sketch of the system with the heat source, heat sink and damaged central region.

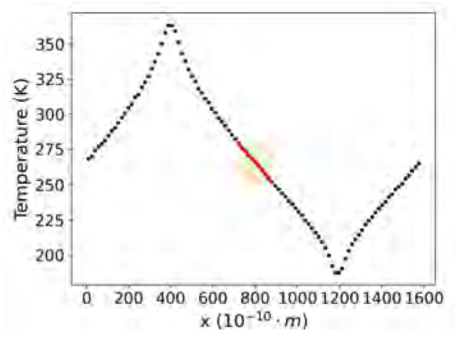


Figure 2: Temperature profile of perfect system

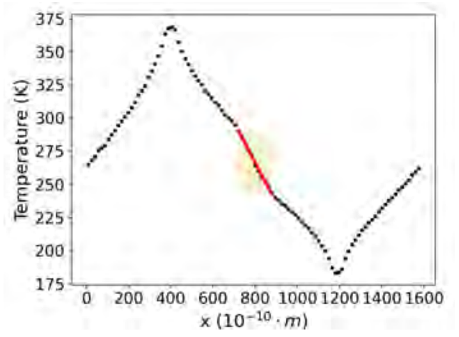


Figure 3: Temperature profile of damaged system

accordingly. Suppose we start the system directly from 600 K at the first steps of the simulation. In that case, the volume changes drastically, leading to unrealistic stress and tension in the simulation box. To avoid these outcomes, we slowly heat the system to the desired temperature, bringing the pressure fluctuations under control. Apart from these, the NVE step also requires longer times for the temperature to converge. Under these circumstances, the thermal conductivity of Tungsten at 600 K is calculated for several different damage levels. The results are tabulated in Table 2 and plotted, as shown in Figure 4 (b). Even though one can see that the thermal conductivity of the undamaged system is higher than the others, a general correlation between thermal conductivity and the number of defects after recombination cannot be established. Nevertheless, the deviations in thermal conductivity for damaged systems are minor. Therefore, they show an approximate value for damaged systems, around 5.80 W/mK.

Table 2: Thermal conductivity at 600 K

Defects		
Initial	Final	κ(W/mK)
0	0	6.91
6960	1248	5.32
8692	1612	5.89
28976	1622	6.09
31667	1723	5.57
26189	1962	6.03
9239	2130	5.87
9900	2130	5.81
10422	2222	5.96
30478	2708	6.00

Conclusions

The first thing interpreted is that the thermal conductivity of Tungsten decreases after radiation damage. Likewise, we observed a decrease in thermal conductivity as the temperature increased. It is also observed that due to the randomness involved in creating damage and recombinations of defects back into perfect regions during thermal equilibration processes, the correlation between applied damage and thermal conductivity cannot be established. Thus, the interpretation of the results for materials design is quite limited. To expand this study, the simulations can be repeated several times to generate a statistical result for individual temperatures. Moreover, specific analysis methods can be designed to identify how the applied damage changes the total energy of the system. As a result, the nature of recombination can be understood, and significant conclusions can be drawn. Once these relations are established, the thermal conductivity of irradiated Tungsten can be investigated at higher temperatures.

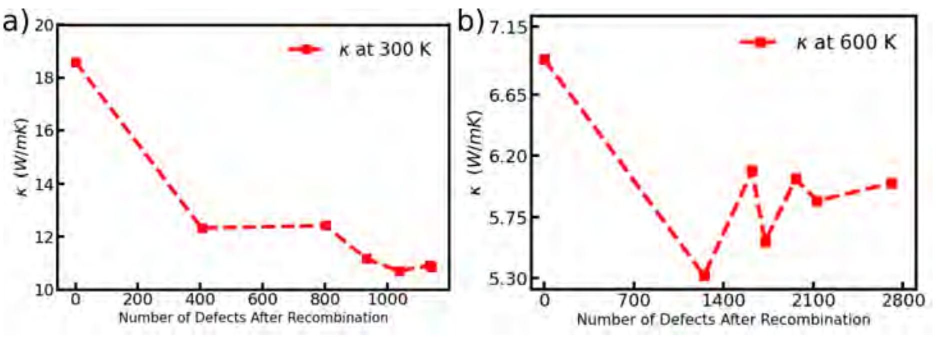


Figure 4: Thermal conductivity for different numbers of defects at 300 K (a) and 600 K (b).

Acknowledgments

We want to thank our supervisors, Mary Kate Chessey and Julio Gutiérrez Moreno, for their active support, motivation, and their non-stop availability. Also, we would like to thank Mervi Mantsinen for having us in the BSC Fusion Group and giving us this opportunity. Moreover, we would like to thank the PRACE SoHPC program for this excellent experience!

References

[1] Ikeshoji, T. and Hafskjold, B. (1994). Non-equilibrium molecular dynamics calculation of heat conduction in liquid and through liquid-gas interface

[2] Wirnsberger, P.; Frenkel, D. and Dellago, C. (2015). An enhanced version of the heat exchange algorithm with excellent energy conservation properties

[3] Derlet, P. M. and Dudarev, S. L. (2020). Microscopic structure of a heavily irradiated material

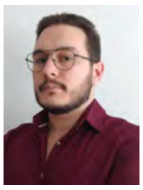
PRACE SoHPCProject Title
Fusion reactor materials: Computational modelling of atomic-scale damage in irradiated metal Official title of the project

PRACE SoHPCSite
Barcelona Supercomputing Center, Spain

PRACE SoHPCAuthors
Arda Erbasan, Middle East Technical University, Türkiye
Dominik Freinberger, Vienna University of Technology, Austria

PRACE SoHPCMentors
Julio Gutiérrez Moreno, BSC, Spain
Mary Kate Chessey, BSC, Spain
Mervi Johanna Mantsinen, BSC, Spain

PRACE SoHPCContact
Arda, Erbasan, Middle East Technical University
E-mail: arda.erbasan@metu.edu.tr
Dominik, Freinberger, Vienna University of Technology
E-mail: e11708140@student.tuwien.ca.at



Arda Erbasan

Dominik Freinberger

PRACE SoHPCSoftware applied
LAMMPS, Ovito

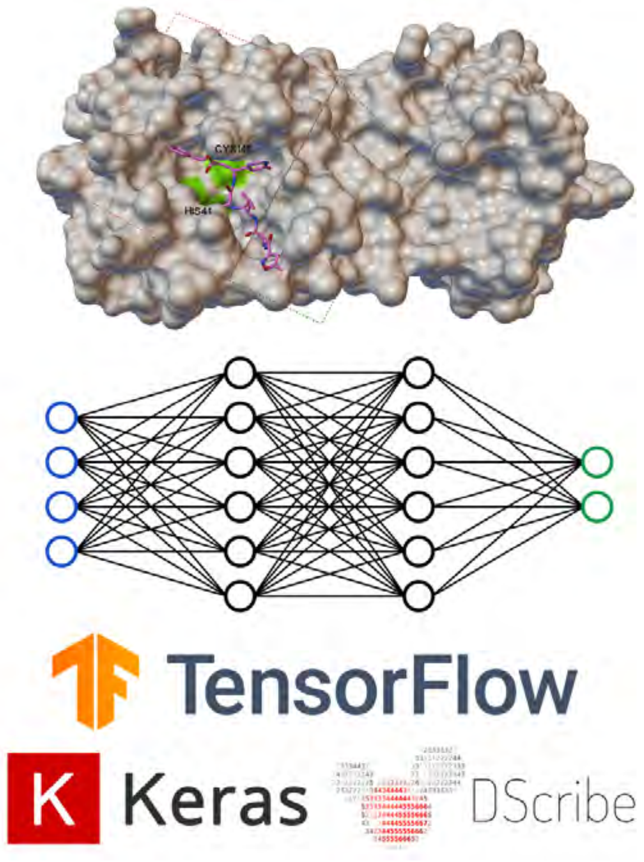
PRACE SoHPCMore Information
https://fusion.bsc.es/

PRACE SoHPCAcknowledgement
This project was developed within the framework of the FusionCAT project (001-P-001722) co-financed by the European Fund for Regional Development of the EU within the framework of the 2014-2020 ERDF Operational Program of Catalonia, with the support of the Generalitat of Catalonia.

PRACE SoHPCProject ID
2202

Discovering new drugs with the advanced computational tools

ANNs to search for potential drugs for COVID-19



Gabriel Cathoud & Tanya Kushwahaa

Presently, developing new drugs to act against COVID-19 is of utmost importance. The main objective of this project is to apply artificial neural networks (ANNs) models to evaluate which chemical compounds could be used as medicinal drugs to treat this disease. This is done by computing docking scores which represent the strength of a drug-target interaction (DTI). The potential drugs will open new pathways towards the future search for methods to combat coronaviruses and would help in preventing any future outbreaks.

Developing new drugs to cure diseases is a long and complex process. The goal is not only to find an effective new medicine but to also guarantee that the product is safe. To assure that, a series of rules and protocols have been developed over the years, such as the guidelines from the International Council for Harmonization (ICH) and the regulations from the Food and Drug Administration (FDA).

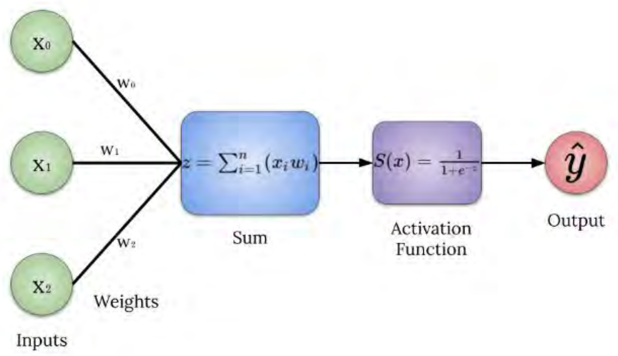
Most drug development processes are very time consuming and cost demanding. It takes on an average 13 years to discover a new drug, and deliver it to the market (Nag et al., 2022). Since, this is a very expensive and exhaustive pipeline, screening wrong candidates to advance steps can result in enormous loss of capital and time. For this reason, the pharmaceutical industries have put strong efforts to develop better pre-clinical techniques, and therefore, ensure that appropriate drug candidates are selected for the next phases.

After identifying a target that is related to the disease, the next step is to find chemical compounds that could be used as drugs. This is a very challenging task, since the number of potential drug-like compounds is estimated to be between 10^{23} - 10^{60}, while the number of all compounds that have been synthesized so far is on the order of 10^8 (Nag et al., 2022).

In the early stages of the pharmaceutical industry, the drug development was carried out by the fundamental trial and error method through a series of testing, validations, and synthetic procedures. With the advent of computational tools such as artificial intelligence (AI) and Deep Learning (DL), most of these practices have evolved.

How modern computational tools can accelerate drug development?

AI is a branch of computer science that deals with the development of algorithms and techniques to solve complex problems which typically require human intelligence. Over the years, AI has been advancing, and used in many fields including finance, transportation, manufacturing, health care, education, etc. Although AI was introduced much earlier to solve biomedical problems, its limitations had prevented widespread acceptance and application in drug development. After the arrival of DL, many of these limitations were overcome (Kaul et al., 2020). AI-based techniques are now used to screen and predict properties of candidate compounds. Apart from property predictions, AI can also been used in other areas of drug discovery like de novo designing of chemical compounds and proteins (Nag et al., 2022).

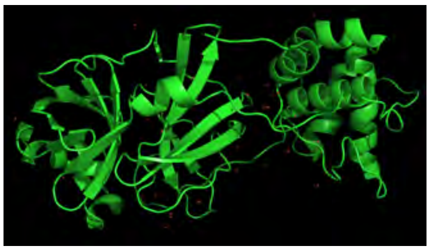


Although AI is a very broad field, and modern approaches are focused more on machine learning (ML). In simple words, ML is defined as the capability of a machine to identify patterns from the available databases. The machine tries to recognize underlying relationships from data in a so called "learning" phase, and apply that information to a similar problem. An advanced version of ML is DL which consist of developing algorithms to create an artificial neural network (ANN) that can learn and make its own decisions, just like neurons in the human brain.

ANNs (hereafter, NNs) are a group of artificial neurons that takes the available data as input, passes it through one or more hidden layers, and direct it through an output layer. Each neuron corresponds to an activation function. These activation functions can perform non-linear transformations on the inputs and send the predictions to the next layer. The process is repeated until a certain number of epochs is achieved. The predictions obtained are compared with the expected values of the problem. In regression problems, typically a mean squared error (MSE) is used as the loss function to evaluate the performance of the model. The training procedure depends on different parameters namely, number of neurons, number of hidden layers, learning rate, types of activation functions, optimizers, bias and weighting functions.

When it comes to selecting candidate compounds, an important parameter in drug development is the docking score. The docking score estimates how good a drug-like compound interacts with a given target, and is therefore, a measurement of the Drug-Target Interaction (DTI). The docking score is very useful for virtual screening because drug candidates can be ranked based on their affinity with the desired target. The docking score is mostly computed using a technique called docking. Docking requires a lot of information about the target and the ligand, which every so often is not available or can also be computationally costly. Another alternative to study the DTI is to use molecular dynamics (MD). However, it is well known that MD simulations require huge amount of computational resources. To overcome these challenges, recent approaches using ML has been adapted leading to fast development of this field.

Although there are vaccines, already developed by the pharmaceutical companies (for e.g., Pfizer by BioNTech), which provide substantial protection from the virus. There is still a lack of effective medicines that can be used to treat already infected patients. This is because a vaccine is effective for the prevention of a virus, whereas a medicine is used to treat a person who is already infected. For this reason, there is a need to develop new medicines as a curative treatment.



With AI and ML, the process of drug development could be accelerated. AI, NNs, and ML have been persistently applied to predict the outbreak (Niazkar & Niazkar, 2020), and also to calculate the molecular docking scores in search of COVID-19 inhibitors (Cheke, 2020; Gerçek et al., 2021). In this study, ANNs are used to search for candidate compounds for the Severe Acute Respiratory Syndrome Coronaviruses 2 (SARS-CoV-2), also called COVID-19. The target selected was the 3CL^{pro}SARS-CoV-2 (6WQF) protease, which is a protein responsible for the virus replication. The goal is to create a predictive model for training using a data set of molecular compounds. After obtaining a good model, this could be used to predict the docking score of new molecules that are promising candidates to act against COVID-19. Based on the docking scores predicted with the ANN models, the actual compounds which can act as inhibitor for covid-19 can be selected.

The importance of molecule descriptors

When using NNs for drug development, a large number of molecules are given as input to train the network. There are different ways to represent a molecule, such as using graphs, in which each edge is a chemical bond, and each node is an atom. One such example is a molecular descriptor which transforms a molecular structure into a mathematical representation.

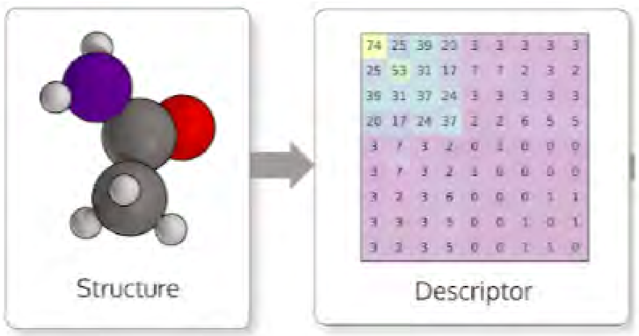
When dealing with AI-based techniques the representation needs to be carefully chosen, since it can have a huge impact on the model predictive performance. To improve accuracy and speed of the models, the descriptors must satisfy some requirements. For example, they need to be invariant to transformations that does not change the properties of a molecule, such as changes in atom indexing, translations, rotations, and reflections. Other requirements are continuity and differentiability with respect to atomic coordinates, and suitability for regression. A descriptor should also be general, i.e., it should be able to encode any atomistic system. Finally, the descriptor should be simple and fast to compute.

There are essentially two types of descriptors: local descriptors and global descriptors. The former set of descriptors include smooth overlap of atomic positions (SOAP) and atom-centered symmetry functions (ACSF) suitable for describing a local environment such as forces and chemical shifts around the atoms of a molecule. Coulomb matrix (CM), bag of bonds (BoB), many body tensor representation (MBTR) all comes in the later category preferable for predicting global properties of a molecule.

Table 1: Best optimization obtained for the following hyperparameters of the different molecular descriptors.

Descriptor	NN Topology				MSE
	No. of neurons	No. of layers	Activation function	Optimizer	
Coulomb matrix	[200, 200, 100, 10]	4	Adam	Softplus	0.03
Many body tensor representation	[100, 50, 5]	3	Adam	Softplus	0.50
Atom-centered symmetry functions	[50, 50, 50, 50]	4	Adam	ReLu	1.04

As an example, the Coulomb matrix mimics the electrostatic interaction between the nuclei and the atoms of a molecule following the equations given below:



$$M_{i,j} = \begin{cases} 0.5Z_i^{2.4}, & for\, i = j \\ \frac{Z_i Z_j}{|R_i - R_j|}, & for\, i \neq j \end{cases}$$

The Coulomb matrix is a square matrix in which the diagonal elements represent the interaction of an atom with itself whereas the off-diagonal elements represent the Coulomb repulsion between nuclei *i* and *j*. On the other hand, descriptors like ACSF can be used to represent the local environment near an atom by using a fingerprint composed of the output of multiple two- and three-body functions that can be customized to detect specific structural features.

Methods

The inputs for the NNs were created using a Python package called DScribe (Himanen et al., 2020) which contain several molecular descriptors. In the project, both local and global descriptors were utilized from the package namely, CM, MBTR, SOAP, and ACSF. Moreover, Python libraries specifically built for machine learning and deep learning, TensorFlow (Abadi et al., 2015) and Keras (Chollet et al., 2015) were constantly used to develop NN models during the project. TensorBoard, a tool for providing the measurements and visualizations during the machine learning workflow, was also used. The NN training is done using the COVID-19 data set of Bucinsky et al. (2022).

To fasten the processes of creating molecular descriptors and NN training, the computer cluster of Slovak Academy of Sciences (SAS) and the HPCFS cluster of Slovenia was utilized. The computing centre of SAS has 128 CPUs containing 1 core per socket. The HPCFS cluster has a total of 64 CPUs containing 16 cores per socket.

Tuning the parameters of the descriptors

Each molecular descriptor has its own set of parameters. Hence, the parameters of the descriptors were tuned to obtain mathematical arrays, which best describes the investigating molecules. These parameters could also affect the size of the descriptors vector and, therefore, impact the input layer of the NNs. In the case of ACSF, for instance, the type of symmetric function and the cutting radius are examples of parameters that can be modified.

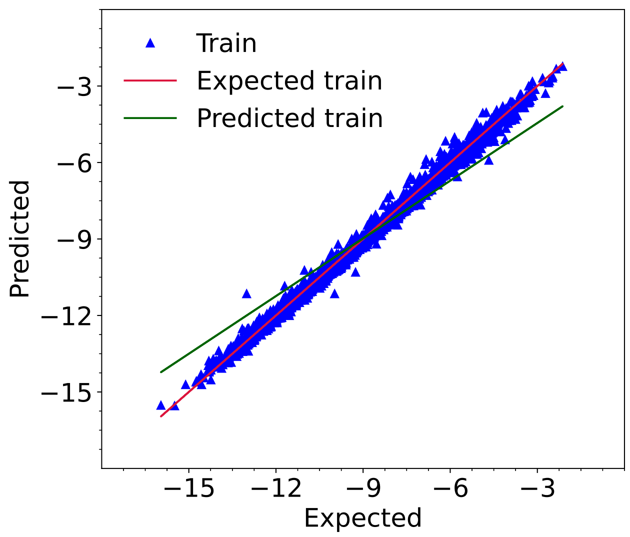


Figure 1: The training process of the data of the candidate molecules against COVID-19. The expected and predicted energy values are shown in red and green, respectively.

Tuning the hyperparameters of the NN models

In the study, feed-forward neural network (FFNN) was used. Even though the topology of FFNNs is simple, they still have numerous hyperparameters to tune in order to obtain the finest model suitable for the COVID-19 data. The efficiency of the training depends strongly on the tuning of the hyperparameters. These hyperparameters are the number of hidden layers and their number of neurons, activation function (AF), types of optimizers for the back propagation, batch size, and the method to initialize the weights and biases of the AFs.

Each one of them was tuned separately, keeping the parameters of the descriptors fixed. The tuning was done for a set of values of each parameter. To ease the process, `for` loops were run in TensorBoard which provides an interactive visualization of the results from all the runs. The final training was done with the model having best hyperparameters for both the descriptors and the neural networks.

Results

The best hyperparameters obtained from the tuning of the neural networks is provided in Table 1. Three or four number of hidden layers, and the optimizers *Softplus* and *ReLu*, seem to return the best optimization for the data set used in this study. Furthermore, the activation function *Adam* seems to be the best activation function for all the descriptors. The least MSE is obtained for Coulomb Matrix (0.03), followed by MBTR (0.50) and ACSF (1.04). Considering the simplicity and the lowest MSE, the Coulomb matrix appears to be the best

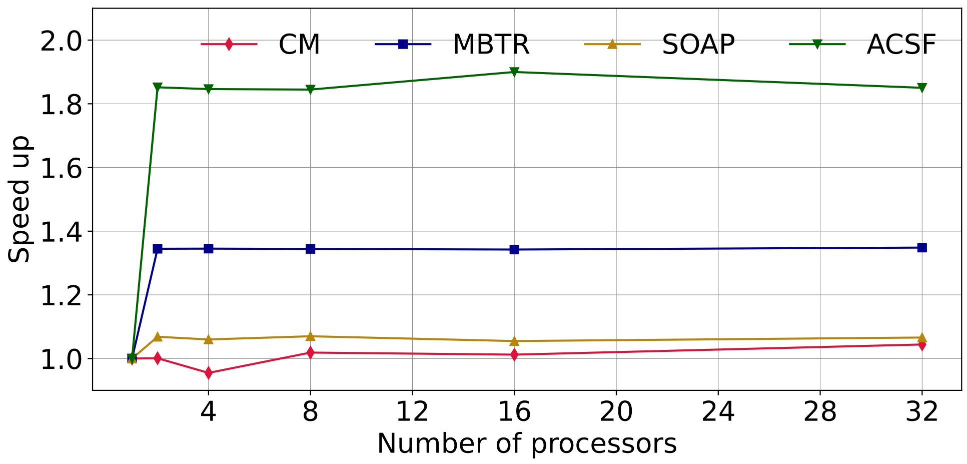


Figure 2: The speed-up curve for molecular descriptors: Coulomb matrix (red), many body tensor representation (blue), smooth overlapping of atomic positions (yellow), and atom-centered symmetry functions (green).

molecular descriptor to describe candidate compounds to be used as drugs for COVID-19.

Final training was carried out after tuning all the hyperparameters i.e., of the descriptors and of the NNs. In Figure 1, the predicted values obtained from the final training in which molecules were represented using Coulomb descriptor is plotted with the expected values. The values obtained for the R^2 and MSE were 0.997 and 0.03, respectively. Clearly, there is some spread in the data points, and the final model obtained is not perfect.

The speed up of the creation of the molecular descriptors was also investigated. The speed up is defined as the ratio of the time required by a program when run sequentially (i.e., using one processor) to the time required by a program when run parallelly using 'n' processors, $Speed\ up = \frac{T_{Serial}}{T_{Parallel}}$. Different number of processors (n=1, 2, 4, 8, 16, 32) were used for the creation of the descriptors. Figure 2 shows the plot of the speed up curve with speed up computed for the CM, MBTR, SOAP, and ACSF descriptors. The parallelization works well for the complex descriptors ACSF and MBTR, and their creation can be speed up by $\times 2$.

Conclusions

Even though the final models obtained from the CM, MBTR, and ACSF descriptors does not show the best performance compared to other studies. They still reasonably predicts the docking score of the candidate compounds. Coulomb Matrix which is the simplest molecular descriptor produced satisfactory results in the training. All in all, its non-complexity and effective results makes it a good candidate among other molecular descriptors to produce adequate preliminary results during the drug development process using AI. The speed-up curve of the complex descriptors (e.g., ACSF and MBTR) occurs to be less parallelizable as expected which is maybe due to a problem in the architecture of the IBM HPC server and the implementation of the DScribe library.

Due to lack of time, all the hyperparameters could not be tuned thoroughly and, therefore, the parameters obtained from tuning were not optimal. Additional work is required to study all the descriptors and their hyperparameters in detail, and to obtain better models for drug development.

Given so many parameters, it is unlikely that with a small number of trainings, and a simple grid search one can obtain the optimal NN architecture and the best parameters for molecular descriptors. Moreover, huge amount of time and energy will be required to compute all the possible solutions. Evolutionary computation may work as a solution for this which could be explored in the future.

References

Abadi, M., Agarwal, A., Barham, P., et al. 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, software available from tensorflow.org

Bucinsky, L., Bortňák, D., Gall, M., et al. 2022, Computational Biology and Chemistry, 98, 107656

Cheke, R. 2020, Eurasian Journal of Medicine and Oncology

Chollet, F. et al. 2015, Keras

Gerçek, Z., Ceyhan, D., & Erçağ, E. 2021, Turk. J. Chem., 45, 704

Himanen, L., Jäger, M. O. J., Morooka, E. V., et al. 2020, Computer Physics Communications, 247, 106949

Kaul, V., Enslin, S., & Gross, S. A. 2020, Gastrointest. Endosc., 92, 807

Nag, S., Baidya, A. T. K., Mandal, A., et al. 2022, 3 Biotech, 12, 110

Niazkar, H. R. & Niazkar, M. 2020, Global Health Research and Policy, 5, 50

PRACE SoHPCProject Title
Neural networks in chemistry – search for potential drugs for COVID-19

Gabriel Cathoud

PRACE SoHPCSite
Slovak University of Technology, Bratislava, Slovakia

Tanya Kushwahaa

PRACE SoHPCAuthors
Gabriel Cathoud, Tanya Kushwahaa, University of Coimbra, Portugal University Bourgogne Franche-Comté, France

PRACE SoHPCMentor
Dr. Marián Gall, Slovak Technical University, Slovakia

PRACE SoHPCCo-Mentor
Dr. Michal Pitoňák, Comenius University in Bratislava, Slovakia

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

PRACE SoHPCSoftware applied
Python libraries: TensorFlow, DScribe

PRACE SoHPCMore Information
https://www.tensorflow.org/
https://singroup.github.io/dscribe/latest/

PRACE SoHPCAcknowledgement
We would like to express our sincere gratitude to our project Mentor Dr. Marián Gall and Co-mentor Dr. Michal Pitoňak for guiding us throughout the project, and providing their assistance and support whenever needed. We would further like to thank PRACE for this memorable experience. Part of the calculations were performed in the Computing Centre of the Slovak Academy of Sciences, and HPCFS cluster, Slovenia.

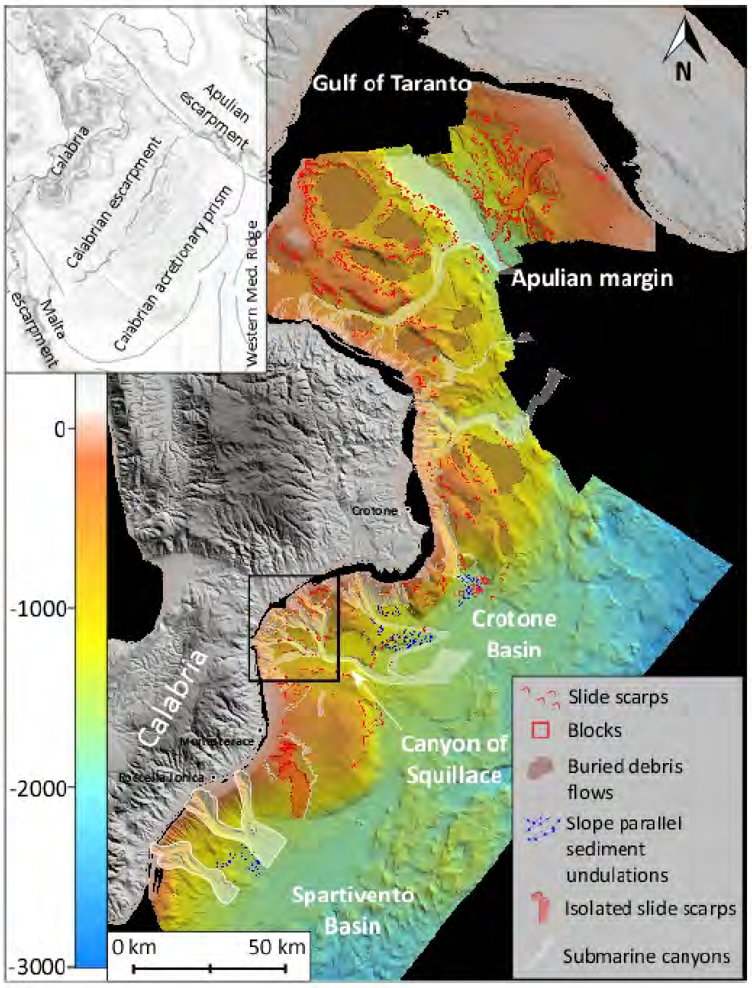
PRACE SoHPCProject ID
2203

Automated Extraction of Satellite Bathymetric data by Artificial Intelligence strategies

Satellite images and Machine Learning for extracting bathymetry

Román Alarcia Pérez

Estimating bathymetry data from satellite reflectance imagery using two algorithms, Machine Learning and a band ratio algorithm. Study of how different features can be used to approach the problem.



The sea depth data is called bathymetry and it has several and useful application in many fields such as navigation, dredging planning, environmental management or aquaculture. Currently less than 20 % of the ocean´s seabed has been mapped, and the scientific community is doing a big effort to complete and increase the resolution of these maps. Traditionally, the water depth is measured by using in situ sensors like vessel-based multibeam sonar or active non-imaging airborne lidar bathymetry (ALB). However, these methods are constrained by access, speed, deployment cost and efficiency in shallow waters.

Bathymetry can also be extracted by satellite techniques, and although it does not have the accuracy of sonar or lidar it can provide wide swaths, lowcost repeated coverage, and easy access to remote areas. There are many satellite techniques for extracting bathymetric data, most of them are still very experimental, and in this project, we will study the use of machine learning (ML) and Caballero´s algorithm (ref. 1).

Material

Satellite reflectance L3 with a of 60 m resolution images of the shallow waters of the Ionian Sea in the south of Italy will be used to train the ML and generate results. A multitemporal set of reflectance images from 2020 to 2022 obtained from satellites Sentinel 2A and 2B will be processed using python and the cluster of CINECA Marconi 100 (ref 2.).

We have studied more than 80 km of coastline from Le Castella up to Torreta, in Calabria, at the south of Italy. Results will be compared with real bathymetry data from the European Marine Observation and Data Network (EMODnet). We limited the study to shallow waters up to 20 m depth.

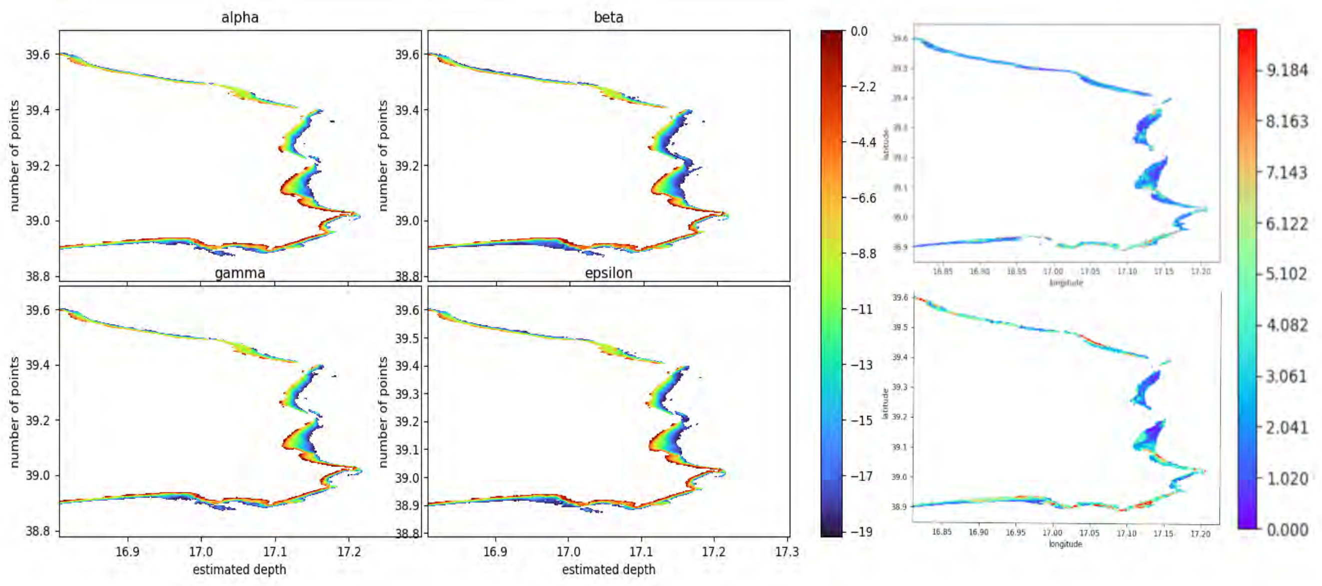
Caballero´s algorithm

In the visible range of wavelength, colours red and green are more adsorbed in the water than blue. Caballero et al. proposed a magnitude, combining the reflectance of these colours, called pseudo-depth (pSDB) which increases proportionally to the depth (SDB). pSDB is defined as the coefficient, between the logarithm of the reflectance blue over green or blue over red (See eq. 1)

$$SDB = m_1\, pSDB - m_0 \qquad (1)$$

$$where\ \text{pSDB} = \frac{\ln\left(\text{n}\pi Rrs\left(\lambda_i\right)\right)}{\ln\left(\text{n}\pi Rrs\left(\lambda_j\right)\right)} \qquad (2)$$

Here, n is just a positive constant that we add to be sure that the logarithms are always positive, in our case we used 10^6. m_o and m_1 are regression constants to estimate the SDB from the pSDB. λ_i is the blue band and λ_j the green or red band. With our data we had 2 blue bands one red and another



At the left bathymetric maps obtained for each one of the 4 pSDB usign Caballero´s algrithm. At the top right corner, a map of the absolute error per pixel of the estimated bathymetry using Caballero algorithm. At the bottom right corner, a map of the absolute error per pixel of the estimated bathymetry using RF.

green, so we could do 4 pSDB combinations.

In total, more than 100 satellite images of the same area have been considered to solve the problem. Images with few points or low accuracy have been removed. For each one of the rest of images we have calculated the SDB doing a linear regression with the pSDB. Using the more accurate images we did an average getting the final SDB. We repeated this process for each one of the 4 pSDB combinations that we had obtaining the results.

Machine learning

The same problem has been studied with a random forest regressor of 200 decision trees. Random Forest (RF) has been used because it can give us the importance of each input, called permutation importance. It is calculated by changing the weight that RF gives to each feature and checking how this affects the final result. When the feature weight change, the more its importance, the more it affects to the final result. 20 different estimators have been compared with this method, 8 colour reflectance bands, and 12 ratios (pSDB) calculated as the equation 2 between the blue bands and the rest of the bands (See table 1). For training the ML we used the 80 % of the data of only one image, then we tested the method with other images.

The most important colour bands for RF are blue, green, red and near infrared. Regarding the ratios, the most important are ratios between blue and red, blue and green and blue and near infrared. Using only these features give as good results as using the 20 inputs. So, the rest of inputs can be removed without deteriorating the final result and saving computational time.

Notice, that for RF near infrared is an important band, but Caballero´s algorithm doesn´t consider this band. Near infrared just penetrates a few millimetres into the water, so it can´t be used to estimate depth. However, this band is used to do noise surface corrections such as light reflections. This fact proves that when working in a new problem, ML can gives us information of which variables are more useful to solve it, so we can understand better new problems.

Maps of the absolute error per pixel have been done to compare the results using Caballero´s algorithm and ML. In general RF gives us better results with a lower error.

Conclusions

Two methods have been tested to measure the depth in shallow waters, RF and Caballero´s algorithm. We have proven that both methods are useful and generate acceptable results, being RF the best one in terms of accuracy. RF importance of 20 features have been studied for the problem. We have proven that ML can be a good first approach for new problems since it gives us what variables are more useful.

References

[1] Caballero et al. (2019) Retrieval of nearshore bathymetry from Sentinel-2A and 2B satellites in South Florida coastal waters. Estuarine, Coastal and Shelf Science 226 (2019) 106277

[2] S.S.J.D. Mudiyanselage et al. (2022) Satellite-derived bathymetry using machine learning and optimal Sentinel-2 imagery in South-West Florida coastal waters, GIScience & Remote Sensing, 59:1, 1143-1158,

PRACE SoHPCProject Title
Automated Extraction of Satellite Bathymetric data by Artificial Intelligence strategies

PRACE SoHPCSite:
CINECA, Italy.

PRACE SoHPCAuthor:
Román Alarcia Pérez, UBFC, France.

PRACE SoHPCMentor
Silvia Ceramicola, TS, Italy

Román Alarcia Pérez

PRACE SoHPCContact
Román, Alarcia Pérez, UBFC
Phone: +34 633 226 871
E-mail: romanalarciaperez@hotmail.com

PRACE SoHPCSoftware applied
AI techniques in python.

PRACE SoHPCMore Information
https://summerofhpc.prace-ri.eu/automated-extraction-of-satellite-bathymetric-data-by-artificial-intelligence-strategies/

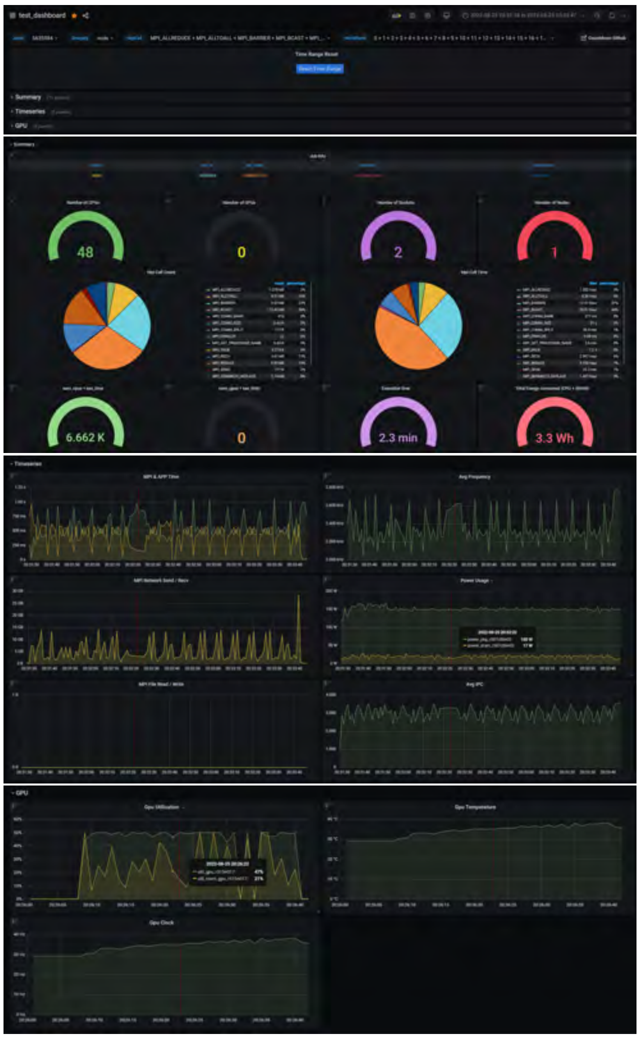
PRACE SoHPCProject ID
2204

A dashboard for on-line assessment of jobs execution efficiency

Job execution dashboard for HPC

Pierre POLLET

Combining a low level acquisition program with a database to create a beautiful dashboard monitoring jobs runtime. In other word, visualize in real time execution of parallelized program.



Applications (called Jobs) used in supercomputers consume a lot of energy which is a limiting factor for scalability of these computers. These programs parallelize the computations across multiple "nodes" each distributed across multiples cores of CPUs. And they need to be synchronized during their computations to coordinate with each other. However, such parallelized applications often waste CPUs power during synchronization of the processes. This is where COUNTDOWN comes in! It is a tool which reduces the CPUs frequency (therefore reducing energy consumption) during idle / synchronisation time while trying to be as impact less as possible.

The Countdown library

Parallelized jobs use MPI functions to communicate with each other during runtime. In order to access those data / metrics we need a profiler to collect data about MPI. MPI Profilers already exists from IBM or Intel for instance, but they come at a cost: They lead to considerable perturbations of the performance of the program and they only report the data collected at the end of the job and not during runtime.

While Countdown is a library used to reduces energy consumption, it is also an MPI Profiler. This library developed and used in CINECA, can also produces timeseries report, report some data each second while the job is running.

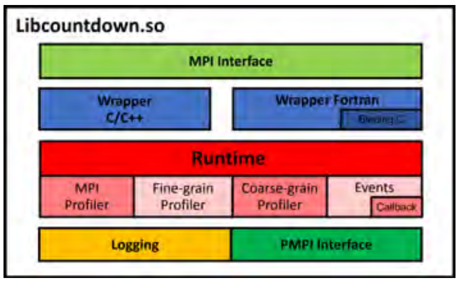


Figure 1: Countdown shared library logical view

This is the library I used to collect all the required information I needed for the dashboard.

Countdown with the Examon DB

Examon is a distributed and scalable monitoring infrastructure with a database. By using Countdown with Examon it is possible to send MPI Profiling information during runtime to the Examon database. Then it became only a matter of design to create a beautiful dashboard using the open source, state of the art, interactive data visualization, web application: Grafana.

But Countdown wasn't fully equipped at first to send all the data that it collected. After a good understanding of how it works, I was able to extend it functionality to send to Examon (via the MQTT protocol) the data collected at the prologue, during runtime, and at the epilogue.

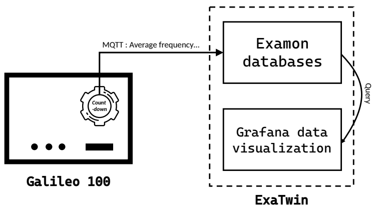


Figure 2: Overview of the architecture

In order to helps users, understand how their programs are running with COUNTDOWN and how much energy they use, my goal is to design a dashboard to monitor programs which are using Countdown and extend Countdown to send all the necessary information about the job.

Design of the Grafana dashboard

Top

At the top of the Grafana dashboard, the user is able to select the Job Id of the job currently running. Then with a little bit of javascript, the Time Range Reset panel set the time range of the entire dashboard to match the job start time and end time (if the job has finished).

Then the dashboard is split into three different section:

Summary

The first part is the summary where all the general information are such as the job name, the partition on which the job is running, the number of cpus, gpus, nodes... And this is also where is the data about MPI function calls. It allows the user to control which MPI calls slow down the program.

Timeseries

Then comes the timeseries part of the dashboard that contains information that Countdown send at each time sample (usually each second).

Gpu

At the end of the dashboard are the GPU timeseries data (if enabled).

Conclusion

I am really glad of what I did during this summer and the results I provided. The Countdown library was a challenging code to encounter, and I think the dashboard is quite nice for a user. I did not struggle much during the whole project except at the beginning when it was hard to understand the architecture of the project and how the countdown library work with other tools.

References

[1] D. Cesarini, A. Bartolini, C. Cavazzoni, L. Benini., P. Bonfà (2018). COUNTDOWN: a run-time library for application-agnostic energy saving in MPI communication primitives.

[2] D. Cesarini, A. Bartolini, C. Cavazzoni, L. Benini., C. Cavazzoni, M. Luisier (2020). Countdown Slack: A Run-Time Library to Reduce Energy Footprint in Large-Scale MPI Applications

PRACE SoHPCProject Title
A dashboard for on-line assessment of jobs execution efficiency

Pierre POLLET

PRACE SoHPCSite
CINECA – Consorzio Interuniversitario, Italy

PRACE SoHPCAuthors
Pierre POLLET, Télécom SudParis, France

PRACE SoHPCMentor
Andrea Bartolini, University of Bologna, Italy Federico Tesser, CINECA, Italy

PRACE SoHPCContact
Pierre, Pollet, Télécom SudParis
E-mail: pierre.pollet54@gmail.com

PRACE SoHPCSoftware applied
Countdown, Grafana

PRACE SoHPCMore Information
Countdown

PRACE SoHPCAcknowledgement
I am thankful to all the mentors and colleagues from the University of Bologna and the CINECA for their help, guidance and support throughout the project and also the EuroHPC REGALE project.

PRACE SoHPCProject ID
2205

Assessment of the parallel performances of permaFoam up to the tens of thousands of cores and new architectures.

Permafrost on parallel

Dimou Stavros, Teber Doğukan

The goals of this project are to build a tool to decompose a mesh in parallel and to analyze the performance of permaFoam in an MPI environment.

Permafrost modeling is a concerning issue that has resurfaced in the main media recently due to global climate change, as it expands both in cold areas and on a general global basis.

Based on the occurrence of numerous non-linearities encountered in the underlying physics, there is a strong need to run experiments on parallel. One of the solvers available regarding the issue is permaFoam, which is built under the OpenFOAM framework. It has been extensively tested, and used for multiple projects like studying a permafrost dominated watershed in Central Siberia, the Kulingdakan watershed (Orgogozo et al., Permafrost and Periglacial Processes 2019[1]) with 500 cores. Apart from that, it has also been tested and analyzed with the use of up to 4000 cores to investigate it's HPC scalability (e.g.: Orgogozo et al., InterPore 2020[2]). Due to the large scales to be dealt with in different projects, it is anticipated that the use of permaFoam with at least tens of thousands of cores simultaneously will be needed.

The goals set during this internship were the following:

Build a pre-processing utility that decomposes a mesh and fields of a case for parallel execution faster than decomposePar.

Make a performance analysis of the permaFoam code, while analyzing critical metrics like execution-time and testing it on supercomputing infrastructures.

The main test case used for the permaFoam analysis, was the 3D OpenFOAM® case which simulates the permafrost dynamics in the Kulingdakan watershed under current climatic conditions (Xavier et al., in prep). As for the supercomputing infrastructures, Olympe Supercomputer (https://www.calmip.univ-toulouse.fr/) was used.

A lifecycle of an OpenFOAM problem

A typical OpenFOAM user goes through three stages to solve their problem: pre-processing, solving, and post-processing. Pre-processing involves creating and organizing the mesh. And, post-processing involves visualizing and examining the result. Let's zoom in on the pre-processing part.

Geometry discretization, meshing, is handled by blockMesh utility so that the created mesh can be read by the different solver for a serial application. For large-scale problems, however, blockMesh alone is not sufficient. Parallel programming for faster computation is required. In order to use parallel programming, the mesh needs to be decomposed into sub-parts for each MPI process. Before a mesh case can be run in parallel though, it has to be decomposed and that operation is done by decomposePar.

decomposePar in a nutshell

decomposePar is a serial pre-processing utility that sets up the mesh for parallel execution. Its job is to split the mesh among processes.

As the mesh gets bigger and bigger, the execution time of decomposePar is increasing. It can be problematic for big-size meshes. The problem we were facing at HiPerBorea was too long execution time of decomposePar. Our solver, permaFoam was designed to be run in parallel. Therefore, we needed a tool that decomposes a mesh of a case for parallel execution faster than decomposePar.

Faster way to decompose

By looking at the source code of decomposePar, it can be seen that the same operations are done for each directory that will be created. For instance, there is a method called writeDecomposition which creates necessary directories and files. In this method, there is a for-loop that loops through each process and does some operations:

```
for (label proci = 0; proci <
    nProcs_; proci++)
{
    // do stuff
}
```

That for-loop contains more than 800 lines of a code block in braces and executes that code block as the number of processes. The source code contains similar types of for-loops in many places.

Adjusting to parallel programming paradigm

Due to the serial nature of decomposePar, as it is mentioned above, some for-loops can be eliminated by parallel programming. Let's start investigating the source code to have an idea of how to replace for-loops with parallel programming.

When we run decomposePar, it creates an object called domainDecomposition. This object has all the information about undecomposed mesh and how to decompose it. It also has two main public methods: decomposeMesh and writeDecomposition. The first one decomposes the mesh and sets up the member fields of the object and the second one creates processor directories and writes the necessary information to files.

The object of domainDecomposition class has members that are used later in the code. Most of these members are initialized in decomposeMesh method. The interesting part about these members is that they hold all the information regarding all processes.

For instance, there is member called procPointAddressing. Its data type is labelListList which is a 2D list of integers. This member holds labels of points for each process. Since decomposePar is running on one process, there is no other way to hold this information separately. But in our case, we are exploiting parallel programming with distributed memory. Because of that, each process can have its domain and members.

The rise of parallelDecompose

If each process is going to have its domain, creating one object is not enough. The number of objects has to be equal to the number of processes. Besides, domainDecomposition class cannot be used anymore. A new, lighter class has to be created. In addition to that, repeated for-loops have to be removed and decomposePar has to be parallelized as much as possible. To satisfy these needs, we have built parallelDecompose utility. It decomposes a mesh and fields of a case for parallel execution in parallel.

We have tested parallelDecompose against 850K cells (100x100x85). For parallelDecompose, 324 cores (18x18x1) have been used on 9 computational nodes. In this mesh case, decomposePar took 1 minute and 8 seconds while parallelDecompose took 32 seconds. By looking at this test result, we can see that parallelDecompose reduced the computation time by at least 50%.

For the total memory usage, decomposePar's MaxRSS value is 1572K while parallelDecompose's MaxRSS is 2779948K. parallelDecompose consumed way too much memory compared to decomposePar. The main reason is that there is a part in the program where we had to gather all the information in one process and scatter that information into all processes. That information is a class member called cellToProc. Every process has this member and its size is proportional to the size of a mesh. Since every process has this member and we are using distributed memory rather than shared memory, the total usage of memory of the individual process is multiplied by the number of processes. That is the root cause of bloated memory.

Performance Analysis

For the performance analysis as mentioned above, the supercomputing infrastructure Olympe was used in order to test the permaFoam behavior while making good use of multiple cores. The performance analysis was conducted for both the demoCase and demoCaseSinusoidalClimateForcing test cases provided by the hydrology repository, where the first case is a simulation of rain water infiltration into a soil layer while the second one illustrates the handling of seasonal variabilities in the boundary conditions.

To conduct this study, the permaFoam source code was edited as multiple time counters and function counters were introduced in order to detect the most time consuming parts of the code. The permaFoam code consists of multiple equations, with the two most important and time consuming being Richard's equation and Heat Transfer equation. Apart from that the code was profiled using VTune profiler as well as LWP : Bull MPI Lightweight Profiler. Both profilers are supported by the Olympe Supercomputer.

Execution Time Analysis

For the first part of the performance analysis, the main goal was to observe how the execution time evolves as the number of cores increase, to evaluate the performance of each equation under these circumstances and to exploit permaFoam's efficiency on different environments. The demoCaseSinusoidalClimateForcing case was tested, while using different number of cores for the time of one whole year (simulation). The tests were conducted while using 1, 36 and 72 cores, corresponding to a serial version of permafoam, and the utilization of 1 and 2 nodes of Olympe.

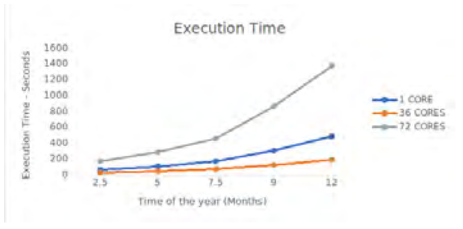


Figure 1: Execution time.

As shown on Figure 1, we can see that the execution time when using 36 cores compared to 1 is much smaller, and specifically the completion of all operations is 2.54 times (154%) faster with 36 cores, showing this way the superiority of using multiple cores. Apart from that we tested the code with 72 cores, dividing the mesh only on the X axis. The mesh we worked on was 50x85, so the domain was over - decomposed quite quickly. As a result, the overall execution time increases radically with the serial version being 2.84 times (184%) faster than the version with 72 cores. Further analysis would have to be made on a bigger mesh.

Concerning the time spent on each equation for different number of processes, the Heat Transfer Equation is the most time - consuming on all cases, in comparison to Richards equation. Specifically with one core, Richard's equation is 1.9 times (92%) faster, while with 36 cores it is 2.2 times (122%) faster. Apart from that the number of iterations conducted by each core is the same as in the sequential version as expected.

Analysis of Linear Solver

Another important part of the permaFoam code is the Linear solver. The Linear solver contains a few iterative operations, and the purpose of this analysis is to observe how those iterative operations evolve depending on the number of cores, as it is suspected that it effects the overall performance. The tests taken were for multiple different number of cores, and were also run for different period lengths, to get a complete understanding of the behavior of the iterative operations.

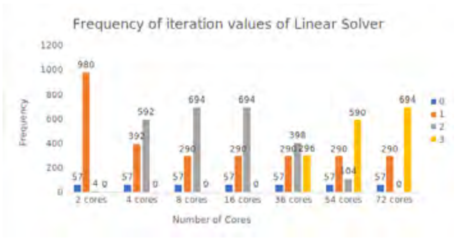


Figure 2: Frequency of iteration values.

It is observed that the total iterative operations are the same independent of the number of cores. However, it is clear from Figure 2, that the number of cores affects the number of iterations occurring on different operations. There is high frequency of bigger number of iterations and low frequency of small number of iterations as the number of cores increases. On the other hand for a small number of cores, the opposite happens with the small number of iterations having a higher frequency. However, for the value of zero iterations the frequency stays consistent and the same, independent of the number of cores used.

Profiling

For the last part of the analysis, we aimed to obtain in depth knowledge on the behavior of the code, and find the most time - consuming parts of it. In order to achieve this goal, we profiled the permaFoam code and the two profilers used for this purpose were VTune and LWP, both supported by Olympe. The profiling was conducted for the duration of a whole year with multiple different cores.

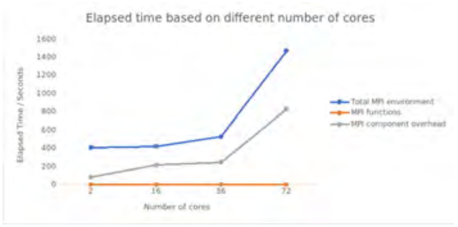


Figure 3: Elapsed MPI time

As expected, Figure 3 depicts that the duration of the Total MPI environment utilized as well as the MPI component overhead that occurs, is increased as the number of cores used increases. In detail, using 2 cores the total MPI environment utilization time was 30% less than that of the 36 cores, and the time caused by the MPI component overhead was 3.01 times less as well. This is more than expected, as the communication between a bigger number of MPI processes will cause greater overhead, and will be more time consuming.

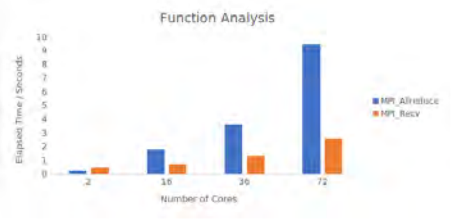


Figure 4: Function Analysis

As shown in Figure 4, the profiling of the code also assisted us in discovering the most time consuming MPI functions and their development as the number of cores increases. The most time consuming functions were MPI_Allreduce and MPI_Recv, both of which dedicate a lot of time for communication with other processes. Specifically MPI_Allreduce spends around 3.6 seconds of total work, while MPI_Recv spends 1.3 seconds respectively, while working with 36 core. All the other functions had a fairly small execution time.

Conclusions and Future Work

parallelDecompose is decomposing a mesh and fields correctly and fast. The tool is ready to use for small and medium-sized meshes. As reported above, it can perform decomposition twice as fast as decomposePar. Before it can be used for big mesh sizes, like hundreds of millions of cells, we need to divide cellToProc class member to each process so that it can only hold its current process' data.

The reason cellToProc could not be divided in this version is that the work performed to assign cellToProc cannot be easily split between the MPI processes because of its intrinsic construction. Making it parallel would involve a complete re-writing of the algorithm, which would need further development.

Concerning the performance analysis, the behavior of permaFoam could be exploited with the use of even more active cores and with bigger meshes to avoid over-decomposition effects. Apart from that it would also be interesting to to experiment on other supercomputing infrastructures.

References

[1] Orgogozo et al., Permafrost and Periglacial Processes 201. Application of OpenFOAM® to permafrost modeling – the permaFoam solver.

[2] Orgogozo et al., InterPore 2020. Numerical modeling of coupled heat and water transport for the study of permafrost dynamics: High Performance Computing simulations for watershed scale analysis.

PRACE SoHPCProject Title
Assessment of the parallel performances of permaFoam up to the tens of thousands of cores and new architectures.

PRACE SoHPCSite
IDRIS, GET-Geosciences Environment Toulouse, France

Dimou Stavros

PRACE SoHPCAuthors
Dimou Stavros, Electrical and Computer Engineering, University Of Thessaly, Greece
Teber Doğukan Software Engineering, Yasar University, Turkey

Teber Doğukan

PRACE SoHPCMentor
Xavier Thibault, Toulouse, France

PRACE SoHPCContact
Dimou Stavros, University Of Thessaly
E-mail: stevedimaras@gmail.com
Teber Doğukan, Yasar University
E-mail: dogukanteber1@hotmail.com

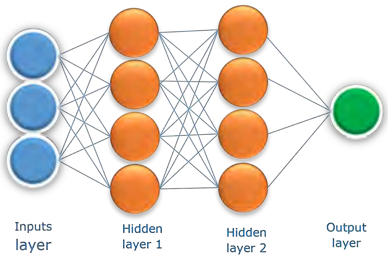
PRACE SoHPCSoftware applied
C++, MPI, OpenFoam, permaFoam, VTune, LWP

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

PRACE SoHPCAcknowledgement
We would like to thank our mentor for the project Mr Xavier Thibault. We are grateful for the support and guidance he offered during this project, as he spent a great amount of time into helping us from the beginning. Apart from that, thank you to PRACE and all the members of the organizing team that lead to this SoHPC being conducted.

PRACE SoHPCProject ID
2207

Mechanical Models with Neural Networks



The concept of neural networks is proving to be a game changer in many research fields, providing answers to complicated problems. This project aimed at approximating results of mechanical models leveraging neural networks and improved performance in term of precision was achieved.

Ahmed Senior Ismail

Machine learning has a significant field of applicability and attractive theoretical formalism. It employs clever formulations and use robust optimization algorithms.
Deep learning is a subset of machine learning that uses neural networks with many layers. Neural networks, for instance, have multiple advantages in terms of performance and flexibility [1],[2]. However, building good performant neural networks requires lots of data and lots of computational resources for training, as a result of intensive computation. Thus, this makes HPC a necessity in order to unravel these caveats.

What is a neural network?

Neural network is one of the most prominent and important machine learning applications. It is a process inspired by the functioning of the neuron in the human brain[3]. A typical neural network is known as input and output system with layered structure. It consists of nodes known as neurons and the connections between them called synapses[4]. It has been demonstrated that neural networks are universal approximators [5]-[8]. Besides, the universal approximation theorem states that neural networks can be used to approximate any kind of continuous function. Besides, the universal approximation theorem states that neural networks can be used to approximate any kind of continuous function

The driving forces

Prior to moving ahead, I would like to highlight the motivation for this project. Modern industrial projects involve various technical stages before production. For instance, the design, the sizing and the certification stage. These stages require building models and running heavy calculations such as finite or discrete elements based on specialized software. However, oftentimes the studied components share similar parameters that makes it possible to group them into various parametric classes. Hence, this makes it possible to handle them using machine learning. The concept of neural networks was proposed in place of the conventional approaches for solving problems related to physical models due to its of its flexibility and universal character of approximation.
Besides, the optimization of this concept sums up as the process of improving the performance of neural networks.
This study focuses on building neural networks to predict the resistant effort or reserve factor of mechanical models. Also, to improve the precision relative to the lowest possible error threshold.
The objectives are; to study the precision of a single perceptron. And to study the precision of neural networks relative to the lowest possible error thresholds for two different mechanical models.

Use Cases

The two mechanical cases considered are linear and nonlinear components. Synthetic datasets have been generated for each case, which means that we had both the size and the quality of the datasets controlled.
The features of these datasets include parameters of the geometry, material properties, and applied forces while the target is a maximum resistance.

Model Building

Artificial neural networks performs better for regression analysis because it has the ability to learn complex relationship between the features and target due to the presence of activation function in each layer.
The models have been built leveraging keras module of tensorflow python package. The training uses dense layer architecture. In addition, the mixed precision policy of tensorflow module was incorporated. To get the most suitable parameters for model training, hyperparameter tuning was carried out using kerastuner module. Lastly, the post process analysis was carried out on best model configurations saved during training, thanks to the callbacks modules.

Models Analysis and Results

As earlier discussed, the purpose is to build and improve the precision of our networks relatively to the lowest possible error thresholds. Here the accuracy is defined as the ratio of errors under a threshold and the total number of samples. While the precision of the models corresponds to the smallest threshold with a perfect accuracy. Below is the mathematical expression of the accuracy.

$$\textbf{Acc} = \frac{\textbf{Card}\{i \in [1,n] || \Delta y_i| < e\}}{n}$$

Application on a linear case: a set of parallel beams which have only the widths as variables and other parameters are constants. The resistant effort can be written as a linear combination of these quantities. For instance, $N_{Rd} = c_1 b_1 + c_2 b_2 + c_3 b_3$.
Where b_1, b_2 and b_3 are the widths of the beams and c_1, c_2 and c_3 are constants.

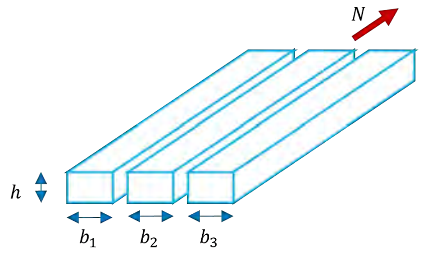


Figure 1: A set of parallel beams that define a linear problem where b_i are variables and c_i are constants.

The first priority here is to explore for maximum precision obtainable for a single perceptron. The major factors that influence precision are optimizer, learning rate, number of iteration and precision. Errors in this case are limited to just training and numerical error because a single perceptron is sufficient to predict results of a linear problem. Also, the datasets used here are synthetic and they are free of noise. After taking a few measures like data preprocessing, tuned learning rate and conducted more calculations, improved results have been obtained.
The accuracy is perfect for relative error threshold down until 10^{-6} which coincides with the single numerical precision. Thus, seeing that has hinted to try double precision. Hence, training with mixed precision, that is, setting different precision for the training and the variables produced improved results. Training with double precision produced better accuracy for low error thresholds, whereas half precision produced very low performance (see figure 3).

Leveraging these measures on multi-layer perceptron though produced a similar performance compared with the single perceptron but evaluating the overall performance in term of mean shows that single perceptron better represents the data. The issue could be that increasing the network capacity tends to overwhelm the linear case, which brings about capacity error. Thus, achieving global minimum in this state might be difficult due to many local minima (see figure 4).

Application on nonlinear components: The problem in consideration here is a single bar with more variables such geometrical and material properties. The major factors that influence precision here is architecture of the neural network, optimizer, learning rate, number of iterations and numerical stability. The errors include training, numerical and capacity errors. Here, the resistance effort is written as, $N_{Rd} = b \cdot h \cdot f_y$.
Where b and h are respectively width and height of the beam and f_y is the limit of elasticity.

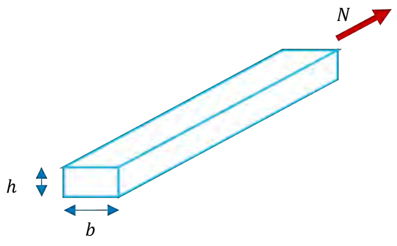


Figure 2: A beam under tension with all parameters as variables. It represents a simplified nonlinear problem.

Networks of single layer but of different widths are examined. The precision achieved is good but bounded around 1% error threshold. Although numerical error seems less significant compared to the other two errors. As expected, bigger architectures yield higher precisions (see figure 5).

Lastly, exploring multi-layered networks appear to yield no real significant improvement to the previously achieved precision around 1% (see figure 6). Increase of capacity can sometimes result in lower performance [9], [10], as more local minima may appear.

Figures

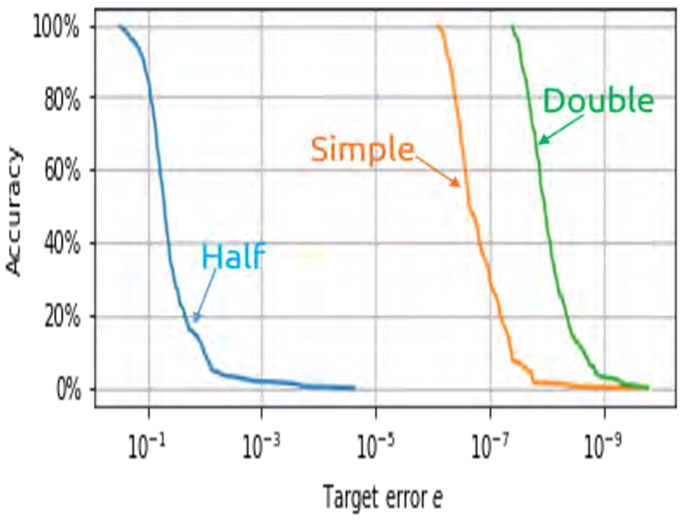


Figure 3: Comparison of different numerical precisions.

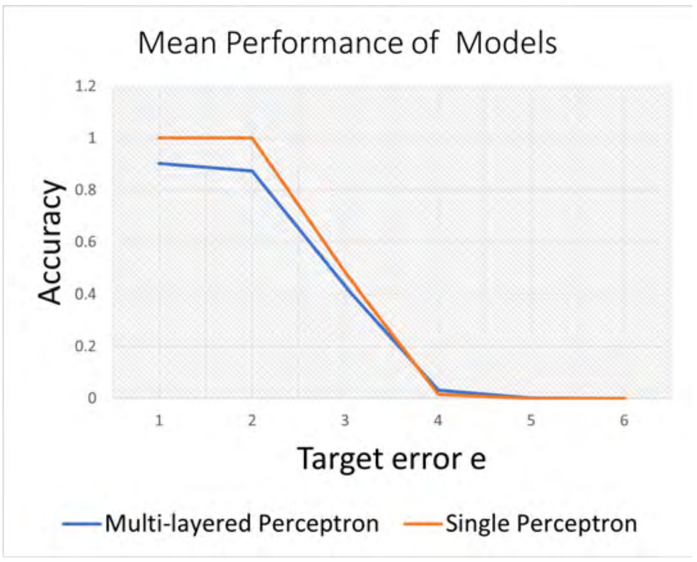


Figure 4: Overall mean performance of MLP versus single perceptron on the linear components

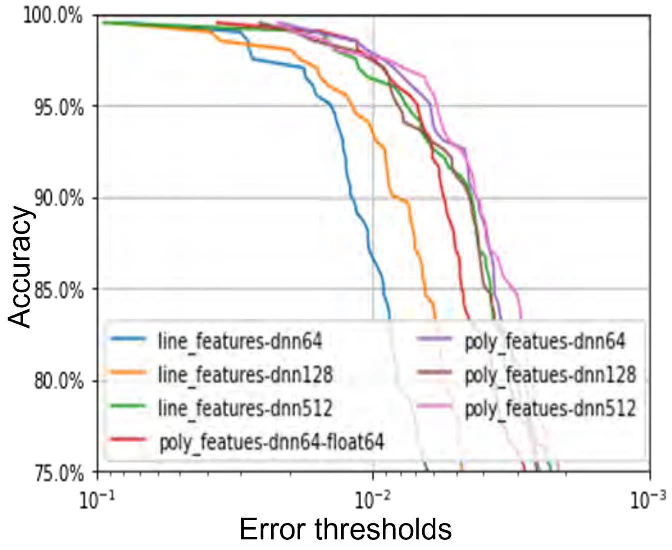


Figure 5: Comparison of the best performant models

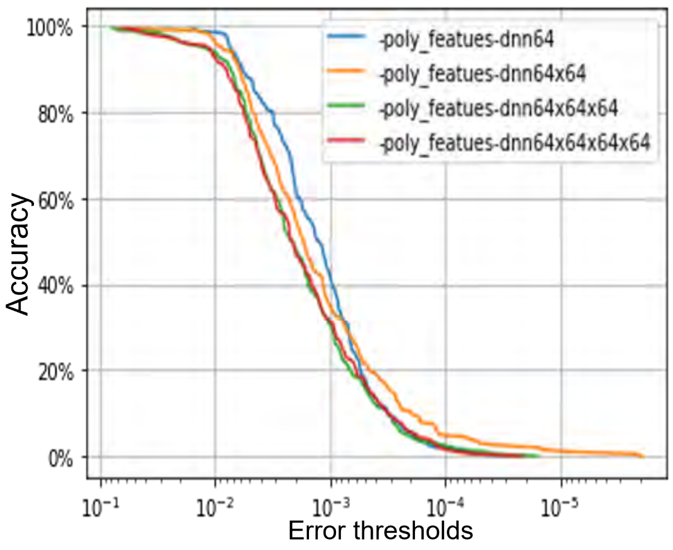


Figure 6: Comparison of the performance of models of fixed widths and different depths.

Conclusion

This project buttresses the use of neural networks as universal approximators to solve mechanical models. In general, other studies for which the components share similar parameters can leverage the concept too. It tends to be more useful when the components to be studied require various levels of complexity, as there is no need for significant modifications.

Although the precision of a prediction appears to be bounded but still good for lots of applications. Finally, high performance computing has been used for the computational demands required of this study. Parallelization using MPI has been implemented to better manage processing ressources and realize statistical approaches.

References

[1] C. Sharpe, T. Wiest, P. Wang, and C. C. Seepersad, 'A Comparative Evaluation of Supervised Machine Learning Classification Techniques for Engineering Design Applications', J. Mech. Des., vol. 141, no. 12, Oct. 2019, doi: 10.1115/1.4044524.
[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.
[3] M.Vasighi, . Artificial Neural Networks: Biological Inspiration Part 1. 2016, doi:10.13140/RG.2.1.3083.6089.
[4] M. Chuks Nwadiugwu, n.d. Neural Networks, Artificial Intelligence and the Computational Brain, doi:2101.08635.
[5] K. Hornik, M. Stinchcombe, and H. White, 'Multilayer feedforward networks are universal approximators', Neural Netw., vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.
[6] D.-X. Zhou, 'Universality of Deep Convolutional Neural Networks', ArXiv180510769 Cs Stat, Jul. 2018
[7] A. Kratsios and E. Bilokopytov, 'Non-Euclidean Universal Approximation', ArXiv200602341 Cs Math Stat, Nov. 2020
[8] Y. EL Assami, B. GELY, n.d. Using neural networks for sizing and certification of mechanical systems: model precision and accuracy. Comput. Struct. Technol. – CST 2022
[9] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, 'Identifying and attacking the saddle point problem in high-dimensional non convex optimization', ArXiv14062572 Cs Math Stat, Jun. 2014
[10] H. Kwak and B.-T. Zhang, 'Understanding Local Minima in Neural Networks by Loss Surface Decomposition', Feb. 2018

Acknowledgements

My profound gratitude goes to the organizers of PRACE Summer of High Performance Computing (SoHPC) for such an awesome opportunity to be part of this project. I would like to thank my lecture Professor Antonio Cicone for making me aware of this program and all other efforts made to ensure that I made it to the program. Also, I would like to thank Mr. Karim Hasnaoui from CNRS, IDRIS laboratory. Furthermore, I extend my gratitude to Capgemini Engineering Company. Lastly, my special appreciation goes to my mentors for their relentless guidance towards the success of this project. I thank you all.

PRACE SoHPCProject Title
Optimization of neural networks to predict the results of mechanical models

PRACE SoHPCSite
Capgemini Engineering Company, France

PRACE SoHPCAuthors
Ahmed Senior Ismail,

PRACE SoHPCMentors
Yassine El Assami, Capgemini, France
Benoit Gely, Capgemini, France

PRACE SoHPCContact
Ahmed Senior, Ismail
Phone: +39 351 583 6187
E-mail: ahmed.ismail.senior@gmail.com

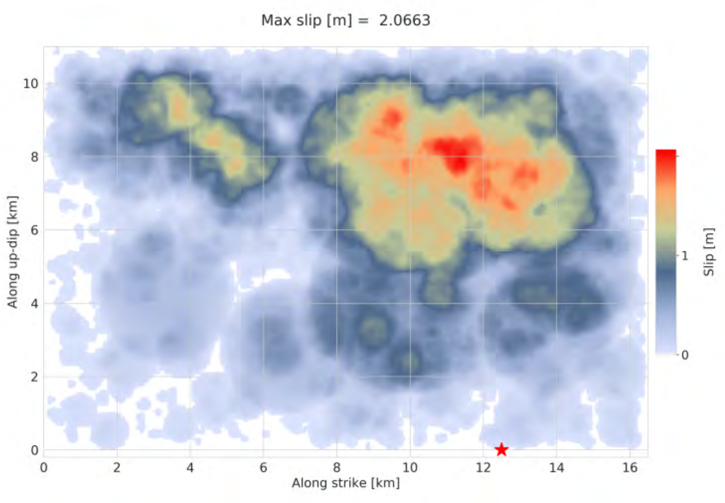
PRACE SoHPCSoftware applied
Python, Paris-Saclay mesocentre, Lecad laboratory(http://hpc.fs.uni-lj.si/)

PRACE SoHPCMore Information
http://mesocentre.centralesupelec.fr/

PRACE SoHPCProject ID
2208

Ahmed Senior Ismail

Stable Earthquake Simulations: a Crash Course



Sara Sandh

High Performance Computing has opened new doors to understanding seismic wave propagation. Numerical solvers can predict complex earthquake scenarios, but performance is limited by the stability of the numerical methods. The implementation of an advanced stability condition can reduce the computation times drastically.

When a volcano erupts or an earthquake occurs, a large burst of energy gets released. The low-frequency energy that propagates through the earth, causing the ground to vibrate, is known as *seismic waves*. Unless you live in a seismic hazard zone, you probably don't spend many hours of your day contemplating how seismic waves propagate throughout the earth. Nevertheless, the interest in understanding seismic wave propagation is simple: by understanding how seismic waves propagate, we can predict the effects on buildings and infrastructure around the source, and therefore minimize or prevent the inflicted damage.

The behavior of seismic waves is difficult to predict due to the irregular geometry and varying material properties of the earth. The application of good numerical methods and High Performance Computing (HPC) offers a competitive solution to overcome such issues.[3]

A good example of this is **SEM3D**, an HPC wave propagation solver based on the spectral element method (SEM) and a Newmark time-stepping scheme. The spectral element method, unlike the better-known *finite* element method (FEM), offers properties that are particularly beneficial for HPC implementations of seismic applications.[2] The time-stepping scheme propagates the solution over time using known solutions of previous time steps to calculate the solution at the next time step.

An ordinary differential equation is obtained when applying SEM to the spatial variables of the wave equation, where the solution to our problem u is a pressure or displacement field depending on the media of the domain (fluid or solid):

$$M\ddot{u} + Ku = F^{ext}$$

Due to properties of the spectral element formulation, the mass matrix M is exactly diagonal. In addition to the low memory requirements that follow, it reduces the complexity of the algorithm and simplifies the parallel implementation.

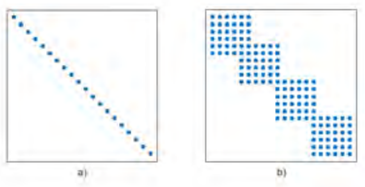


Figure 1: Example of sparsity pattern of a) mass matrix M and b) stiffness matrix K of a spectral element formulation.

Stability - the Performance Bottleneck

The stability of a time-stepping scheme is essential for the reliability of the solver. For intuition, the concept of stability can be compared to a simple real-life scenario: imagine a friend asks you what you will do the next day. Knowing your current occupation, hobbies, and social situation, you can probably give a good, educated guess of what the next day will look like. If your friend instead asks what you will do a year from now, the chance that you can give a good answer decreases. Stability in a numerical time-stepping scheme acts similarly: if the time step Δt is too large, the estimated solution will be poor. As the numerical scheme utilizes the inaccurate data to calculate the solution at the next time step, the error grows with each step and causes instabilities.

The choice of time step Δt is a delicate task - a sufficiently small Δt is required to ensure stability, on the other hand, redundant computations are performed if Δt is smaller than required. To maximize the efficiency of the solver, Δt should be chosen as closely to the stability limit as possible. The classical Courant Friedrichs Lewy (CFL) stability condition can be derived for homogeneous domains and gives an upper bound to the stable Δt based on the wave speed, element size, and order of the SEM. Challenges occur when translating the CFL condition to heterogeneous media - large instabilities are observed when applying the lowest local velocity.[1] In **SEM3D**, the current solution to the heterogeneous stability problem is to multiply the potentially unstable Δt by a user-defined safety factor, often resulting in a very small time step and a great loss of performance.

A more advanced stability condition that extends to the heterogeneous case is based on the maximum eigenvalue of the matrix-matrix product $M^{-1}K$ (matrices obtained in the SEM formulation).[1] Consequently, the eigenvalue-based stability condition computes the optimal Δt for the heterogeneous problem.

The Problem with Matrix Representation...

Similar to large-scale finite element formulations, the spectral element method yields *large* and *sparse* mass and stiffness matrices. As memory usage plays a critical part in the performance of any HPC code, alternative data structures are commonly used to store and represent the sparse matrices. For this reason, **SEM3D** stores the inverse of the diagonal mass matrix M^{-1} as a vector, whereas the stiffness matrix is implemented as a function that calculates a vector of internal forces based on a displacement vector ($F^{int} = Ku$).

Although such matrix representation is highly optimized in terms of storage and performance, it poses a problem as most existing eigenvalue libraries require an explicitly represented matrix to be passed to the eigenvalue solver. Due to the scale and complexity of the existing code, it is not relevant to adapt the matrix representations for this purpose. **SEM3D** furthermore support parallel execution on distributed architectures. The matrices may therefore be divided and distributed over multiple processes according to the adapted input mesh (a full mesh split into N smaller meshes, where N is the number of processes used to solve the problem). The distributed data introduces further challenges as no process holds all the necessary data to calculate the maximum eigenvalue of $M^{-1}K$.

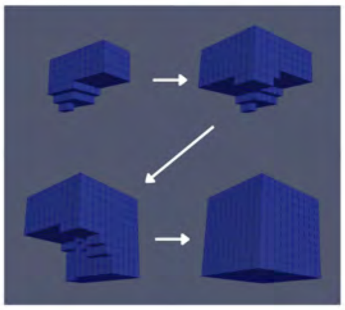


Figure 2: Decomposition of mesh for running **SEM3D** on 4 processes.

ARPACK: the Matrix-Free Eigenvalue Solver

The numerical software library **ARPACK** is specifically developed to solve large-scale eigenvalue problems for sparse matrices without directly acting on the matrix. **ARPACK** implements the Implicitly Restarted Arnoldi Method (IRAM) which, like many other iterative eigenvalue methods, uses the matrix-vector product $Av = w$ to approximate the desired eigenvalue or eigenvector of matrix A.

The most important aspect of the **ARPACK** library is the *reverse communication interface*, an alternative to passing the matrix to a solver. The solver only requires the user to implement a function that performs the matrix-vector multiplication $Av = w$, which is called between each iteration (see pseudocode).

```
! Normal solver that require
! explicit matrix representation
call EigenvalueSolver(A)

! Reverse communication interface
do
  call ARPACKIteration(v, w)
  if (convergence .or. error)
    exit
  else
    call MatrixVectorProduct(v, w)
  end if
end do
```

Listing 1: Pseudocode to demonstrate the reverse communication interface of ARPACK.

The decoupling of the matrix from the solver allows for alternative matrix representations, making the library suitable for HPC applications such as **SEM3D**, for which the representation of matrices often deviates from standards such as 2D arrays, coordinate lists, or compressed row storage.

The parallel version of **ARPACK**, **PARPACK**, supports distributed memory implementations and is particularly suitable when handling a parallel decomposition of matrices, such as in **SEM3D**. Extra care must be taken to the duplicated data on shared faces.

Implementing the Matrix-Vector Product

The matrix-vector product $Av = w$ indirectly lets the matrix A work on the data used in IRAM. For stability of the wave propagation problem, the matrix in question is $M^{-1}K$. For the sequential case, the implementation is straightforward: as K is not explicitly represented, the input vector is passed as a displacement to calculate the internal forces, i.e. $F^{int} = Kv$. The internal forces are multiplied by the vector M^{-1}, to obtain the output vector $w = M^{-1}F^{int}$.

When running the code in parallel, several nodes are shared at the interfaces of the partitioned meshes. Applying the sequential implementation result in duplicated data and incomplete calculation of the internal forces. Code to efficiently communicate the global contributions of the local internal forces is already implemented in **SEM3D**, but attention must be given to the handling of duplicated nodes.

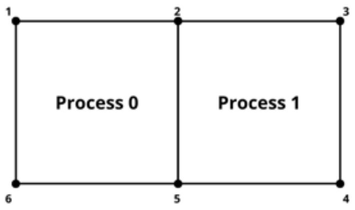


Figure 3: Simple example of two processes and two 2D elements to demonstrate node-sharing of parallel implementation.

In the simple example demonstrated in Figure 3, two processes share two nodes between their adjacent elements. Data such as mass matrix coefficients and internal forces are stored for each *local* element. This results in data related to the shared nodes, i.e. node 2 and node 5 in Figure 3, to be duplicated on both processes, storing the same values. The full local matrices must be used to calculate the internal forces and correct matrix-vector product of the system as a whole:

Process 0 computes:

$$\begin{bmatrix} M_1 \\ M_2 \\ M_5 \\ M_6 \end{bmatrix}^{-1} \cdot \begin{bmatrix} F_1^{int} \\ F_2^{int} \\ F_5^{int} \\ F_6^{int} \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ w_5 \\ w_6 \end{bmatrix}$$

Process 1 computes:

$$\begin{bmatrix} M_2 \\ M_3 \\ M_4 \\ M_5 \end{bmatrix}^{-1} \cdot \begin{bmatrix} F_2^{int} \\ F_3^{int} \\ F_4^{int} \\ F_5^{int} \end{bmatrix} = \begin{bmatrix} w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}$$

PARPACK requires a partitioning of the input and output vectors that don't contain duplicated elements, therefore only the process of the lowest rank will pass its shared nodes to the output vector. The remaining processes will pass a vector of size *#local nodes* − *#shared nodes*. In the example in Figure 3, this would mean that process 0 passes a vector containing elements w_1, w_2, w_5 and w_6 whereas process 1 passes a vector only containing elements w_3 and w_4. The shared data from the input vector is communicated from the lowest ranked process to the higher ranked processes, for the higher ranked processes to reconstruct their full local input vector necessary to calculate the correct matrix-vector product for the next iteration step.

Advanced Stability Condition - is it Worth the Trouble?

The motivation to use an advanced stability criterion lies in the increased performance obtained when fewer iterations of the time-stepping scheme are required. This purpose is defeated if the computation time of the advanced stability condition exceeds the time saved on the reduced number of time-stepping iterations.

The efficiency of applying an advanced stability condition must be discussed keeping in mind two key aspects: firstly, the number of iterations depends not only on Δt, but also on the total simulation time, meaning that the additional overhead from solving the eigenvalue problem will not be as beneficial for very short simulations as for longer simulations. Secondly, as the initially used safety factor is set by the user for each simulation, the gained performance is highly dependent on the user's ability to select a suitable safety factor (a tricky task, one should add).

Despite the above-mentioned aspects, the advanced stability approach showed very promising results. The calculated Δt would typically be around twice the magnitude of other Δt's calculated using the CFL-condition and an appropriately selected safety factor. The advanced stability condition would therefore typically result in improved simulations that require roughly half the number of time-stepping iterations to obtain the same result as the CFL-condition with applied safety factor.

The observed computation time for the eigenvalue problem was *significantly* smaller than that of the remaining solver - in many cases even negligible. Considering the typically conservatively chosen safety factors, the advanced stability condition showed very promising results in terms of overall performance.

Acknowledgements

I would like to express my gratitude to my two amazing mentors, Filippo Gatti and Régis Cottereau. While providing great (and necessary) support, explanations, and pep talks, they let me explore my own solutions and contribute with new ideas and suggestions.

I am also grateful for all the researchers and PhD students I met at LMPS, giving me interesting new perspectives and making every lunch and coffee break a fun adventure!

Lastly, I would like to thank my site-coordinator Karim Hasnaoui for providing great support leading up to my arrival in Paris, and of course, PRACE for organizing this interesting and eye-opening experience.

References

[1] Régis Cottereau and Ruben Sevilla. Stability of an explicit high-order spectral element method for acoustics in heterogeneous media based on local element stability criteria. *International Journal for Numerical Methods in Engineering*, 116(4):223 – 245, October 2018. Publisher: Wiley.

[2] Dimitri Komatitsch, Seiji Tsuboi, and Jeroen Tromp. The spectral-element method in seismology. In Alan Levander and Guust Nolet, editors, *Geophysical Monograph Series*, volume 157, pages 205–227. American Geophysical Union, Washington, D. C., 2005.

[3] Sara Touhami, Filippo Gatti, Fernando Lopez-Caballero, Régis Cottereau, Lúcio de Abreu Corrêa, Ludovic Aubry, and Didier Clouteau. SEM3D: A 3D High-Fidelity Numerical Earthquake Simulator for Broadband (0-10 Hz) Seismic Response Prediction at a Regional Scale. *Geosciences*, 12(112), March 2022. Publisher: MDPI.

PRACE SoHPCProject Title
Implementation of an Advanced Stability Condition of Explicit High-Order Spectral Element Method for Elastoacoustics in Heterogeneous Media

Sara Sandh

PRACE SoHPCSite
CentraleSupélec, France

PRACE SoHPCAuthor
Sara Sandh, Vienna University of Technology, Austria

PRACE SoHPCMentor
Filippo Gatti, CentraleSupélec, France
Régis Cottereau, Laboratoire de Mécanique et Acoustique, France

PRACE SoHPCContact
Filippo Gatti, CentraleSupélec
Phone: +33 (0)175316198
E-mail:
filippo.gatti@centralesupelec.fr

PRACE SoHPCSoftware applied
SEM3D, PARPACK, Fortran 90, MPI

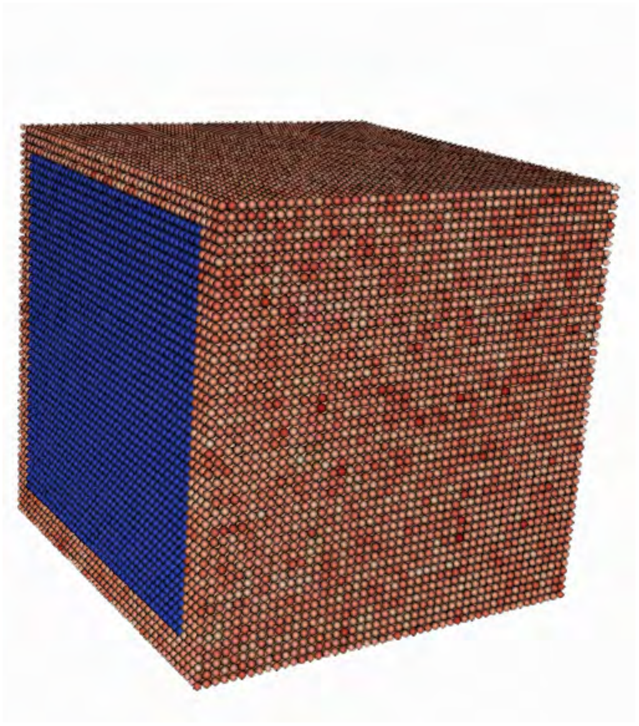
PRACE SoHPCMore Information
Spectral Element Method: Research Paper
ARPACK-NG: Git Repository
SEM3D: Research Paper (1), Research Paper (2)

PRACE SoHPCProject ID
2209

Post-processing optimization

Julen Expósito

We can model interaction between solar wind and space objects, but at the cost of producing large dataset of physical quantities. This project aims to implement post-processing routines for those quantities directly in the simulation code.



When running large simulations, it is not uncommon to have to shop a large amount of data. Usually this data is post-processed to get only the information relevant to a particular study, but this post-processing can be costly and storing the original raw data can be a problem. Why not do the post-processing directly in the simulation, using the power of parallel computing and supercomputers for optimisation and only storing the information you really need?

That is what I'm doing in this project with global simulations of interaction between the solar wind and planetary magnetospheres. This system is defined by the value of the electric and magnetic field in the whole space, as well as the position, velocity, mass and charge of all the particles interacting in it. The code I have worked with, iPIC3D,[1] has already implemented some routines that provide information of interest, such as the current or the pressure tensor in space, but this is insufficient, and researchers are forced to work with the large files resulting from the simulation to calculate what they really need. In this work I have focused on implementing two new routines within the same code, so that they are performed efficiently and in parallel without the need to store much heavier data that would later be used to get the desired information: one to calculate the tensor of temperatures in the whole space, in a coordinate system relative to the magnetic field, and another to calculate the energy spectrum of the particles in the simulation.

With this, future researchers who will use our code to study the interaction between solar wind and planetary magnetospheres will have access to relevant results much faster then before and without having to deal with the huge files that the current code outputs regardless of which specific study you are doing.

The Temperature Tensor

One of the relevant quantities when analyzing a plasma system is the temperature at each point in space. This temperature is related in a straightforward manner under reasonable assumptions to the pressure, which, in general, does not have to be isotropic: that is, it is not, as we are used to, a unique value associated with each spatial point, but a mathematic object representing the forces exerted on a on a differential element of area of the fluid. This is the pressure tensor:

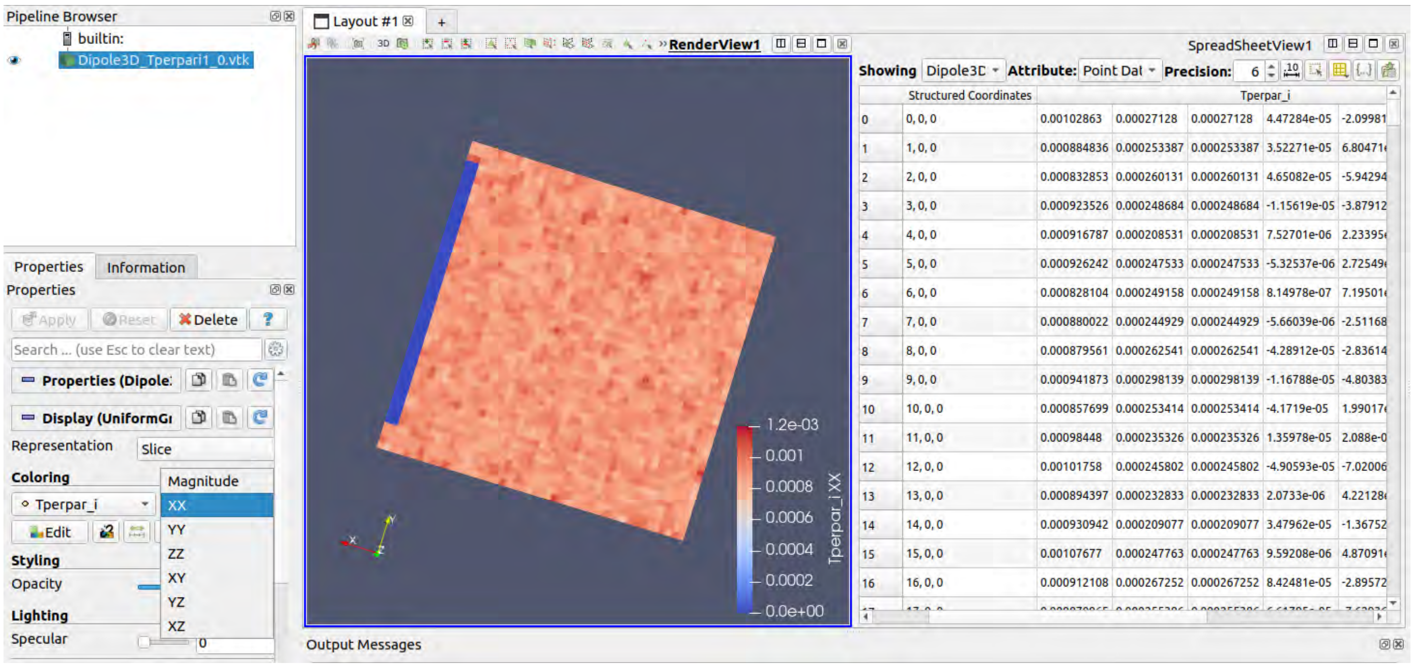
$$\bar{P} = \begin{pmatrix} P_{xx} & P_{xy} & P_{xz} \\ P_{xy} & P_{yy} & P_{yz} \\ P_{xz} & P_{yz} & P_{zz} \end{pmatrix}. \qquad (1)$$

This tensor is what the code gave before this work, creating 6 different files (note that the tensor is symmetric) with the value of each component of the tensor in all the cells of the plasma simulation box. Now, there is a catch here. You want to calculate the temperature tensor from here, but not in the code's cartesian coordinates, but in coordinates relative to the magnetic field in each point of space. This means that you have to take the 6 components of the tensor and realize a change of basis to the matrix to define the tensor in a coordinate system where the principal axis are one parallel to the magnetic field and two perpendicular to it. Note that this coordinates are dependent of the point in space where they are calculated, because the magnetic field is not going to be uniform in general.

Also, it is interesting to have the diagonal components of the tensor in what is called gyrontropic form, where the two pressure diagonal pressure components corresponding to the perpendicular directions are equal. This facilitates the measure of gyrotropy. In the end, you have the following tensor:

$$\bar{P} = \begin{pmatrix} T_{||} & T_a & T_b \\ T_a & T_\perp & T_c \\ T_b & T_c & T_\perp \end{pmatrix}. \qquad (2)$$

This change is computationally expensive, so it is interesting to do it



Representation: Slice
Coloring: Tperpar_i, XX

Showing: Dipole3D Attribute: Point Dat Precision: 6

	Structured Coordinates	Tperpar_i				
0	0, 0, 0	0.00102863	0.00027128	0.00027128	4.47284e-05	-2.09981
1	1, 0, 0	0.000884836	0.000253387	0.000253387	3.52271e-05	6.80471
2	2, 0, 0	0.000832853	0.000260131	0.000260131	4.65082e-05	-5.94294
3	3, 0, 0	0.000923526	0.000248684	0.000248684	-1.15619e-05	-3.87912
4	4, 0, 0	0.000916787	0.000208531	0.000208531	7.52701e-06	2.23395
5	5, 0, 0	0.000926242	0.000247533	0.000247533	-5.32537e-06	2.72549
6	6, 0, 0	0.000828104	0.000249158	0.000249158	8.14978e-07	7.19501
7	7, 0, 0	0.000880022	0.000244929	0.000244929	-5.66039e-06	-2.51168
8	8, 0, 0	0.000879561	0.000262541	0.000262541	-4.28912e-05	-2.83614
9	9, 0, 0	0.000941873	0.000298139	0.000298139	-1.16788e-05	-4.80383
10	10, 0, 0	0.000857699	0.000253414	0.000253414	-4.1719e-05	1.99017
11	11, 0, 0	0.00098448	0.000235326	0.000235326	1.35978e-05	2.088e-0
12	12, 0, 0	0.00101758	0.000245802	0.000245802	-4.90593e-05	-7.02006
13	13, 0, 0	0.000894397	0.000232833	0.000232833	2.0733e-06	4.22128
14	14, 0, 0	0.000930942	0.000209077	0.000209077	3.47962e-05	-1.36752
15	15, 0, 0	0.00107677	0.000247763	0.000247763	9.59208e-06	4.87091
16	16, 0, 0	0.000912108	0.000267252	0.000267252	8.42481e-05	-2.89572

Figure 1: Temperature file opened with Paraview. Below left, the 6 components can be seen as an option for the visualization of their spatial distribution in a 2D slice of the simulation cube. On the right, a table with the component values in each cell of the simulation box.

within the code itself, without having to do it in a later post-processing (which, at the time this project started, only existed serially, which implied a lot of computation time for realistic simulations).

The code works iteratively from given initial conditions, and you can ask it what information you want it to write to files in which cycles. So, I have implemented a new routine that calculates the temperature tensor in the cycles you ask for. In addition, this routine doesn't create 6 different files for each component, but a single file with all the information of the tensor for each point and a format adequate for visualization.

This final result can be seen in Figure 1. Using the data visualization program *Paraview*, the single file allows access to all data at once for comparisons and calculations of all kinds. Each cell in the simulation box has the 6 temperature tensor values associated with it.

The Energy Spectra

Observationally, if we want to study plasma particles in space, what we do is count the number of particles that are in different energy ranges, moving in a particular direction. This means that we are not interested in knowing the positions and velocities of all the particles, but only in knowing the energy spectrum at each point: that is, the probability that a particle has one energy or another.

This information is much less burdensome than the complete information of all particles in a simulation. In the cycles of interest, we could simply calculate how many particles are in each cell of the simulation box and place them in different energy bins. In that way, what used to be hundreds of particles per cell, each with its velocity vector, is now simply a series of numbers indicating the number of particles that fall between different energy ranges chosen by the user. A substantial improvement worth implementing directly in the code for parallel calculation.

This is more complicated than the previous example. Let's think about how the code is designed: the different processes running in parallel have an associated region of space, in which they perform all their computations and which contains certain specific cells. Those cells, in turn, have a certain number of particles inside them.

Each process thus has its own set of particles, and I can make each of them perform a cycle by going through all its particles and putting them in "their place". Let's get down to work.

Suppose we want a power spectrum that goes from, in arbitrary units for the example, energy 1 to energy 5, in bins of 0.5. This means that we want to count the number of particles that have an energy between 1 and 1.5, between 1.5 and 2...

We take particle 1. This particle has a position associated with it in the form of three values: x, y and z. Its position tells us in which cell it is located. Once that cell is found, we take its velocity values in each direction: v_x, v_y, and v_z. Now we have to redo the coordinate change we did with the temperature to take $v_{||}$, $v_{\perp 1}$ and $v_{\perp 2}$, because we want the spectra also in coordinates relative to the magnetic field, and then we use those velocities to calculate the cinetic energy in each direction: $E_{||}$, $E_{\perp 1}$ and $E_{\perp 2}$. Finally, we count that particle in the energy bin where it belongs. For example, if $E_{||}$ is 2.36, you sum 1 particle to the bin corresponding to energies between 2 and 2.5 in the cell where that particles is for the file corresponding to the energy spectra in the direction parallel to the magnetic field.

You do this with all particles and you end with a complete 3D matrix, which each space element having an array of N integer values, where N is the number of established energy bins, and the values correspond to the number of particles in these bins. If you want to count total charge instead of number of particles, just sum the charge of each particle each time instead of 1.

These files are much lighter than the files with all the particle information! And they are much easier to work with. In Figure 2 we can see the energy spectra in the three directions for a certain cell in the simulation box, the total energy spectra and a fit to a Maxwell–Boltzmann distribution, which is the distribution corresponding to par-

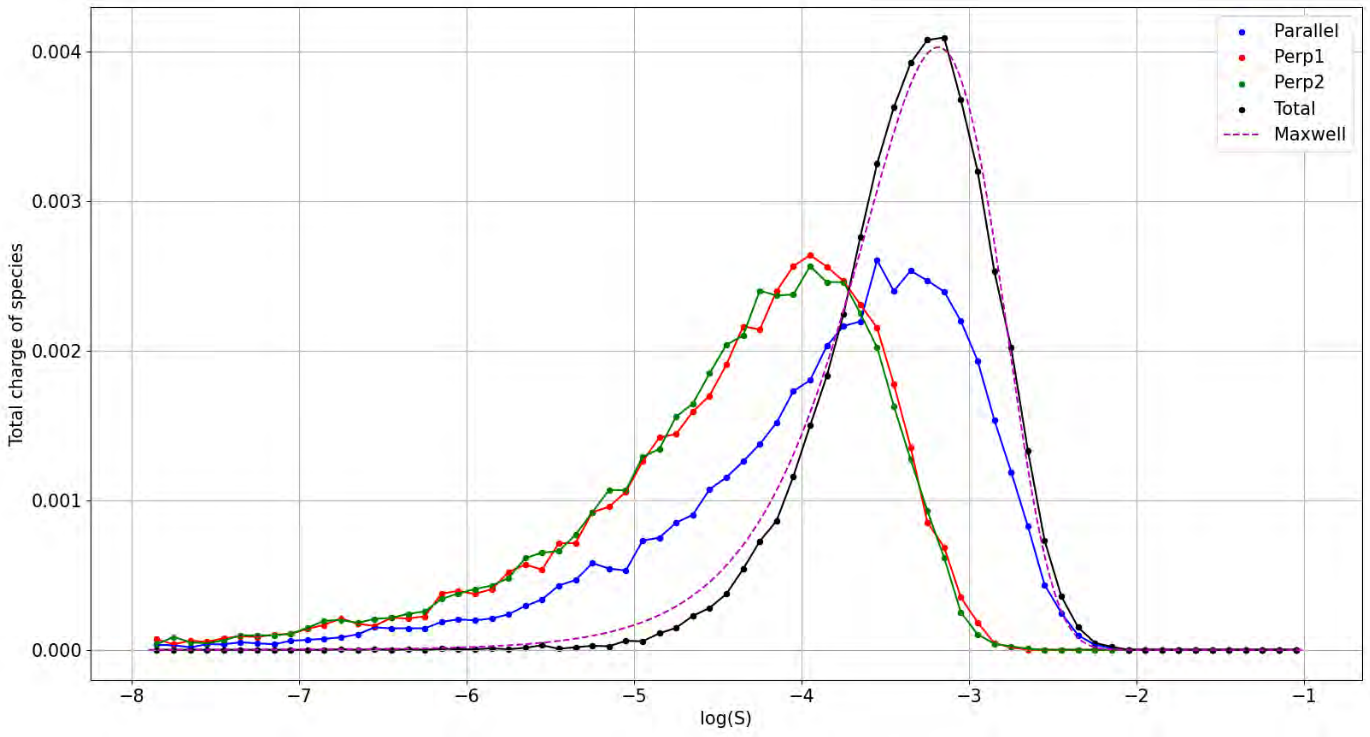


Figure 2: Energy spectra of a random cell of the simulation box for a simulation with uniform magnetic field and uniformly distributed particles with three times initialization velocity parallel to the magnetic field in comparison to the perpendicular directions.

ticles moving freely in a ideal gas, an appropriate approximation of the simulation for which these results were calculated. In this case, the thermal velocity of the particles at initialization in the direction parallel to a uniform magnetic field was three times the velocity in the other two perpendicular directions, and, as it should be expected, we see a more energetic spectra. The Maxwell–Boltzmann distribution fits very well with the total energy, indicating that the spectra is correctly calculated.

The user can select the energy bins and the volume of the cells where the particles are counted, which allows a free selection of the spectrum resolution.

Documentation

When working on collaborative code, never forget that what you write and modify will be part of a titanic project on which many people have worked and will work. Everything you do should be properly commented and documented in a common style that is easy for those who come after you to understand. This is also part of the work.

Both routines are properly commented and tested. Any potential problems are documented, as well as the entire internal process and methodology. As much as it works, it is just as important that people can use it, understand it and modify it in the future without your help!

Conclusions and future work

If we want to make the best possible use of the resources we have, we cannot be storing information that we do not want to use directly. Space is expensive, and working with large files causes many problems. In addition, post-processing of files is usually done with programs and codes that are much less efficient than those that perform the simulations in the first place, which have been developed for years seeking the highest possible efficiency when running on high-performance computers.

These two new routines will allow anyone using the code for the study of plasma in space to have access to relevant information that can be compared almost directly with observational data without having to deal with inefficient post-processing and space problems. The code itself directly provides the relevance information and calculates it by making full use of its parallel design.

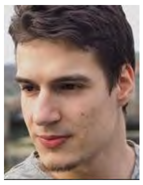
In the future, of course, more post-processing routines can be implemented that continue to speed up and facilitate data analysis and make more optimal use of available resources. The more researchers are able to quickly and efficiently access physically interpretable data, the more and better work will be done using the code.

References

[1] Stefano Markidis, Giovanni Lapenta, Rizwan-uddin (2010). Multi-scale simulations of plasma with iPIC3D.

[2] M. Swisdak (2015). Quantifying gyrotropy in magnetic reconnection.

PRACE SoHPCProject Title
High Performance Data Analysis: global simulations of the interaction between the solar wind and a planetary magnetosphere


Julen Expósito

PRACE SoHPCSite
IDRIS, Laboratoire Lagrange, Observatoire de la Côte d'Azur, Université Côte d'Azur, CNRS, Nice, France

PRACE SoHPCAuthors
Julen Expósito, Spain

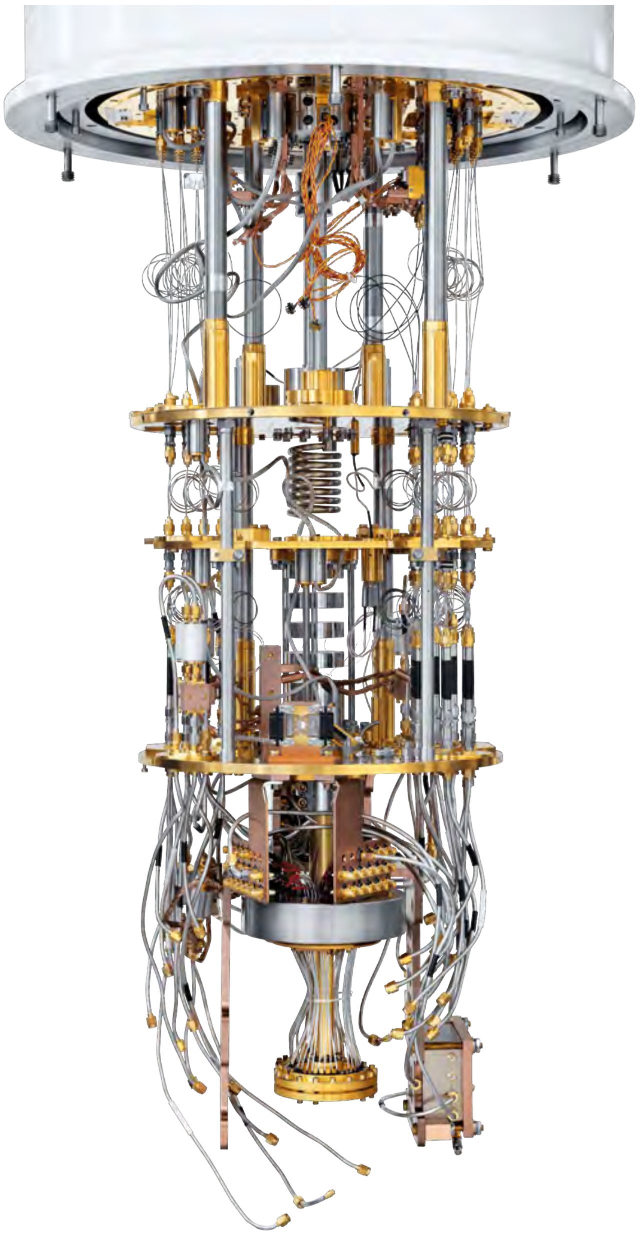
PRACE SoHPCMentor
Pierre Henrišič, Nice, France Federico Lavorenti, Nice, France

PRACE SoHPCContact
Julen, Expósito, ULL
Phone: +34 688 715 525
E-mail: alu0101454177@ull.edu.es

PRACE SoHPCSoftware applied
Virtuoso

PRACE SoHPCMore Information
https://summerofhpc.prace-ri.eu/high-performance-data-analysis-global-simulations-of-the-interaction-between-the-solar-wind-and-a-planetary-magnetosphere/

PRACE SoHPCProject ID
2211



Fundamentals of quantum algorithms

Monika Das, Léo Lassalle

Quantum computing is no longer science fiction. It is already here and evolving quickly. The objective of this project is to understand the main aspects of this growing science. See why a quantum computer is much more efficient than a classical computer on specific problems. We will show it in a concrete way by studying an algorithm called Grover algorithm.

Everyone has heard of quantum computing but in reality few people really know what it means. It is known that quantum computers can solve complex problems quickly. So you must be wondering how it works, or on what kind of problems it is useful?

In this paper we will present and explain the central principles of quantum computing, then we will see several quantum algorithms such as the Bernstein-Vazirani algorithm or the Grover algorithm and finally present a concrete example of quantum algorithm using the Grover algorithm.

Let's start by explaining what differentiates a classical computer from a quantum computer. The power of quantum computing lies in two central principles of quantum mechanics, superposition and entanglement.

Superposition is the ability of a quantum object, like an electron or photon, to be simultaneously in multiple states. Therefore A quantum computer consisting of n qubits can exist in a superposition of 2^n states: from $|000...0\rangle$ to $|111...1\rangle$.

Quantum entanglement is another central idea in quantum physics. It says that multiple qubits can be linked in such a way that the measurement outcome of one qubit's state is correlated with the measurement of other qubit state. The most astonishing thing is that such connection doesn't depend on the distance of the entangled qubits.

Now, let's see some basic concepts of quantum computing.

Bit vs Qubit

A classical computer works with bits, a bit can take the value 0 or 1. It is by creating strings of bits (strings of 0 and 1) that a standard computer communicates.

A quantum computer does not use bits but quantum bits (Qubit), this is what makes all the difference. A quantum bit is not limited to 2 values, the 0 and the 1, but it can also enter in phase of quantum superposition where it combines the 2 states. This is why we represent the quantum bit as a sphere as we can see with figure 1 :

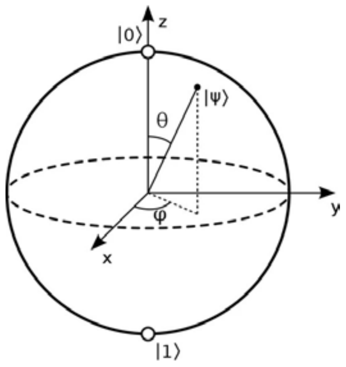


Figure 1: Representation of a qubit by a Bloch sphere.

Here we can see on the z axis the values $|0\rangle$ and $|1\rangle$. All possible points on the surface of the sphere are possible states of the qubit. On the image, the qubit is in the state $|\Psi\rangle$, whose mathematical representation is

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where alpha and beta are complex coefficients with the limitation

$$|\alpha|^2 + |\beta|^2 = 1$$

Quantum Gate

To make a qubit change state, we use quantum gates. Explained visually, these gates allow us to move $|\Psi\rangle$ on the surface of the sphere.
Mathematically, these gates are represented by unitary matrices. Among the most used gates we have the Hadamard gate, as well as the gates allowing to make rotations around the x,y and z axes respectively the X, Y, Z gates.

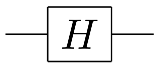


Figure 2: Representation of the Hadamard gate.

The matrice associated with this gate, the Hadamard gate, is

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

By using the different gates together, we can create circuits.

Quantum circuit

A circuit is the equivalent of a computer program for a quantum computer. It is read from left to right. Let's take a simple example with the circuit in figure 3:

Figure 3: Quantum circuit representing a qubit passing through the Hadamard gate.

This circuit has only one qubit q[0] in the $|0\rangle$state at the beginning. Thanks to Hadamard's gate, we make our qubit go from the $|0\rangle$ state to the $|+\rangle$ state. graphically, this means :

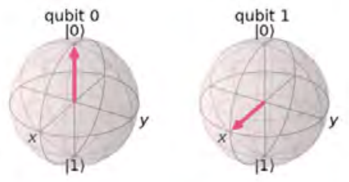


Figure 4: Qubit going from the $|0\rangle$ state (left) to the $|+\rangle$ state (right).

Mathematically, at the end of the circuit our qubit is in the state:
$|+\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$

So in calculating the result, we have a 50% chance of getting 0 or 1. The result of this circuit is given by the histogram in Figure 5.

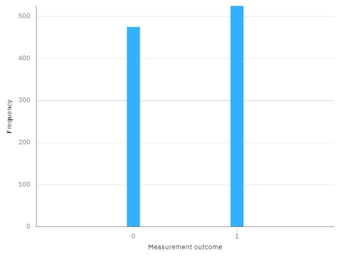


Figure 5: Results of the previous circuit on a 32 qubit simulator with 1000 shots.

When we calculate the result of the circuit, the qubit which is in quantum superposition until then takes the value 0 or 1. This is why when we use a quantum circuit we have to execute it several times, to observe a tendency which will be our result. Here, the state $|+\rangle$ has a 50% chance to tend towards the state $|0\rangle$ or $|1\rangle$.

Quantum algorithms

In this section, we will talk about the quantum superiority on some problems. That is to say the superiority of quantum computers to solve certain type of problems compared to classical computers.

Bernstein–Vazirani algorithm

The Bernstein-Vazirani algorithm is a quantum algorithm to solve Bernstein-Vazirani problem in a faster and efficient way than classical computer. Let us first look into the problem description.

Bernstein-Vazirani problem: Given a function f that maps an input string into 0 or 1,

$$f\colon \{0,1\}^n \to \{0,1\}$$

The function $f(x)$ can be expressed as a bit-wise product of an input string x with an unknown string s

$$f(x) = s \cdot x (\text{mod } 2)$$

. The problem is to find the secret string s.

Classical approach: Classical solution needs to call the function $f(x)$ n times to reveal the full bit string s.

Quantum approach: Using the Bernstein-Vazirani algorithm, we can solve this problem in only one call

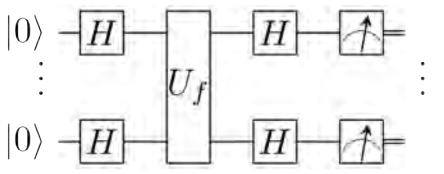


Figure 6: Circuit representation of Bernstein-Vazirani algorithm.

1. First, we create a circuit of n qubits, all in state $|0\rangle$.
2. We apply a Hadamard gate to our n qubits. This allows to create a superposition state.

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

All our qubits are now in the $|+\rangle$ state.

3. Then, we use an oracle, Uf, that change $|x\rangle$ into $(-1)^{f(x)}|x\rangle$ the superposition state is now

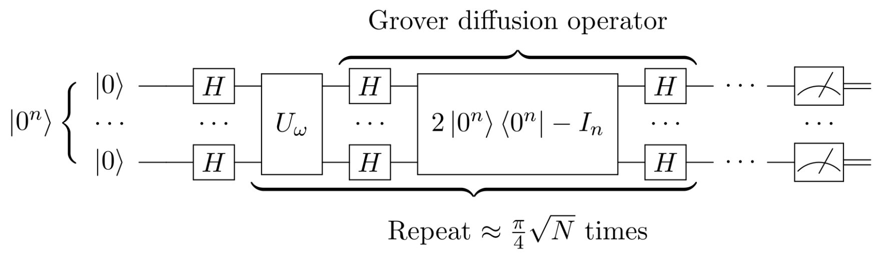
$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)}|x\rangle$$

this means that each qubit tries a string x. If the string corresponds to the searched string s, the qubit changes its state to $|-\rangle$. Otherwise the qubit stays in $|+\rangle$ state.

4. Finally, We apply again Hadamard gate to all qubits. The qubits in $|+\rangle$ state go to $|0\rangle$ state. The qubits in $|-\rangle$ state go to $|1\rangle$ state. Thus we have found the secret string s in a single call.

Grover's Algorithm

Grover's algorithm, also known as quantum search algorithm is a quantum algorithm which can perform a search for unstructured database quadratically faster than any existing classical algorithms.



Grover diffusion operator
$|0^n\rangle$
$|0\rangle$
$|0\rangle$
H
U_ω
$2|0^n\rangle\langle 0^n| - I_n$
Repeat $\approx \frac{\pi}{4}\sqrt{N}$ times

Figure 7: Quantum circuit to implement Grover's algorithm..

Problem statement: Our problem is to locate a unique member w from a list of unsorted database with size N. Let's see how we can do it classically and with quantum computers.

Classical approach: The classical way of searching in such an unstructured database is to look for w in a random manner. Think of looking for a number w in a list of N numbers. If you are lucky enough, you will get this number with $N/2$ attempts. However, in worst scenario, you will need N searches.

Quantum approach: Using the quantum search algorithm, the estimated search needed to find w is $\sqrt{N}$. That is why this algorithm is known to have a quadratic speed-up. Let's go through the step-by-step implementation of Grover's algorithm.

1. The first step is to create a uniform superposition state,

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle.$$

It can be obtained using Hadamard gate $H^{\otimes n}$ on input state $|0\rangle^n$.

2. Apply the oracle operator U_w that introduces a phase flip to the solution state $|w\rangle$.

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

3. Apply the Grover diffusion operator U_s to the state $|s\rangle$ to execute the transformation,

$$U_s = 2|s\rangle\langle s| - I.$$

The Grover diffusion operator in figure 7 does this in 3 steps. First it transform the state $|u\rangle = U_w|s\rangle$ to $|0 \cdot 0\rangle$ by applying Hadamard gate $H^{\otimes n}$. Then it applies the operator U_s and finally it applies the Hadamard gate $H^{\otimes n}$ again to transform back to the state $|u\rangle$ from $|0 \cdot 0\rangle$.

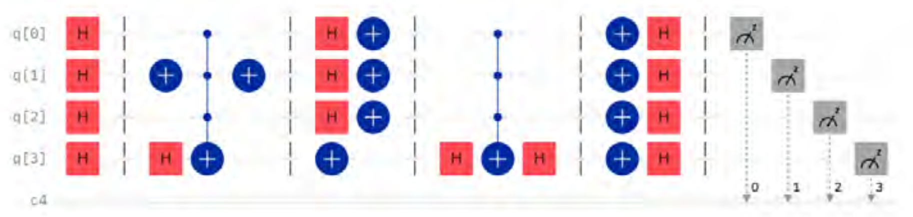


Figure 8: A 4-qubit implementation of Grover search algorithm in IBM's quantum lab for 1 iteration.

4. Repeat the oracle and diffusion operator of steps 2 & 3 for n_{it} times. Number of repetition is expressed as,

$$n_{it} = \frac{\pi}{4\arcsin(1/\sqrt{N})},$$

where $N = 2^n$ is the number of possible input combinations.

5. Measure the qubit states.

The quantum circuit to implement the algorithm is presented in figure 7.

In this work, we implemented Grover's algorithm to search for number 13. We will present the quantum circuit and obtained results in the next sections.

Grover's algorithm to search for number 13

It is clear that a 4-qubit quantum circuit would be required to implement the Grover's search for number 13 or 1101 in binary.

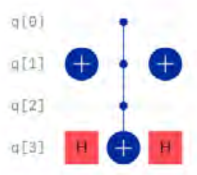


Figure 10: Quantum circuit for Grover's oracle to search for number 13.

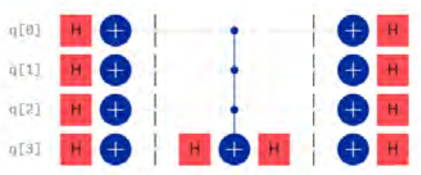


Figure 11: Quantum circuit for Grover's diffusion operator.

The implementation works in two steps. The first step is to build the oracle that maps 1101 into 4-qubit by using a controlled X gate and H gate. The Oracle circuit is quite simple and we only need CNOT and CZ gate for this. Depending on the binary version of the number to search, we need to use controlled X gates where there is a 0 in the binary combination. The target qubit of the CZ gate can be any of the qubits in the circuit. Our oracle circuit in figure 10 uses two Hadamard gates and one

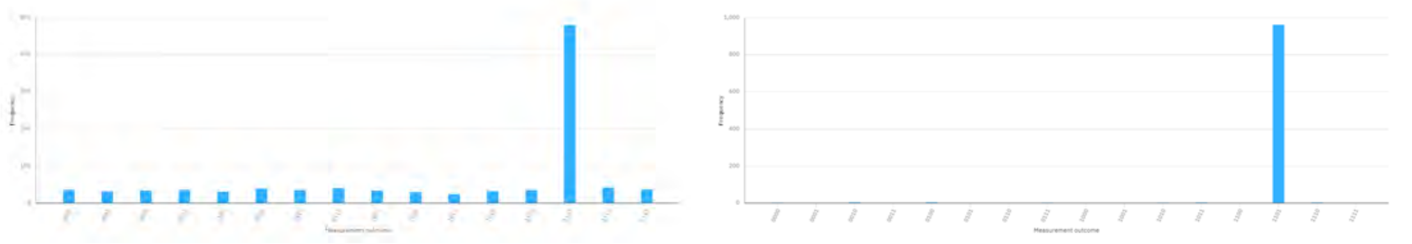


Figure 9: 4-qubit implementation of Grover search algorithm using ibmq_qasm_simulator representing 1 iteration(left) and 3 iterations(right).

X gate to implement the Z gate. Also it uses the last qubit $q[3]$ as the target qubit.

The next step is to build the Diffusion operator as described in step 3. The quantum circuit requires the application of Hadamard gate twice, first at the beginning of the circuit and then at the end. The operator U_s is implemented in between these two applications of H-gates. The circuit is presented in figure 11.

The final step is to combine the oracle and the diffusion operator to construct the full circuit. In some literature, the oracle and diffuser together is named as the Grover's operator. When implemented together, the successive application of two H-gates on $q[3]$ can be omitted as it is equivalent to identity operation.

The quantum circuit in figure 8 contains only 1 iteration of Grover's operator. However, the number of repetition needed for a 4-qubit implementation is 3. In our work, we repeated the oracle and diffuser for 3 times and tested the circuit with both simulator and real quantum computer. We will present and discuss the findings in next section.

Results & discussion

The probability of expected outcome would be highest when the Grover's operator is repeated 3 times. As it can be seen in figure 9, there are still some probabilities for other numbers in the case of only 1 iteration. But for 3 iterations, the expected outcome(number 13) has the highest probability with the other outcomes having nearly zero probability. Therefore, we have obtained the expected result with ibmq_qasm simulator

The circuit has also been tested in IBM's real quantum computer ibmq_quito for 1, 2 and 3 iterations. However, expected outcome is only obtained for 1 iteration (figure 12.

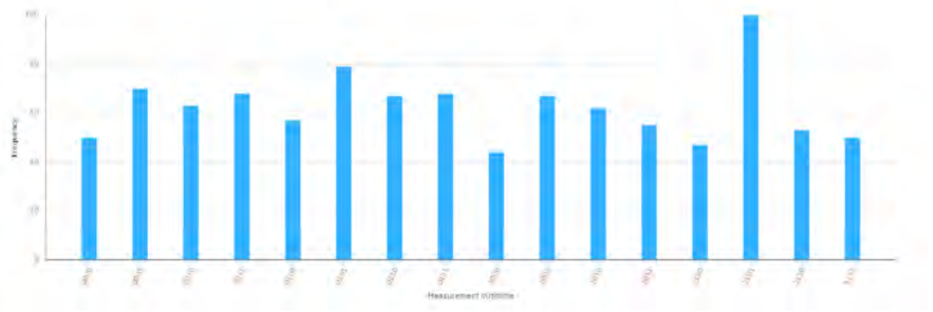


Figure 12: 4-qubit implementation of Grover search algorithm using real quantum computer ibmq_quito for 1 iteration.

As you can see, although 1101 has the highest probability, the numbers also have some probabilities. It is because the real quantum computers are not yet accurate and contain errors. In case of 2 and 3 iterations, the noise amplitude becomes large enough and very random outcomes have been seen.

Conclusion

Throughout the project we have been introduced to many exciting features of quantum computing specially about quantum algorithms. We have also gone through a rigorous exposure to IBM's quantum lab as well as qiskit. Such an exposure enabled us to successfully implement Grover's algorithm. We can safely say that quantum computing is going to provide efficient solutions to future computing problems.

However, we also realized that many of the advancements in quantum computing are yet to come. To solve complex research problems, we need quantum computers with larger number of qubits. The noise in the quantum hardware is also a big issue for accuracy in obtained results. If such limitations are overcome, quantum computing will surely triumph over other computing schemes in upcoming future.

References

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

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

PRACE SoHPCProject Title
Fundamentals of quantum algorithms and their implementation

PRACE SoHPCSite
IT4Innovations, VŠB - Technical University of Ostrava, Czech Republic

Monika Das

Léo Lassalle

PRACE SoHPCAuthors
Monika Das, University of Bourgogne-Franche-Comté, France
Léo Lassalle, Polytech Sorbonne, France

PRACE SoHPCMentor
Jiří Tomčala, IT4Innovations, Czechlands

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

PRACE SoHPCSoftware applied
Qiskit

PRACE SoHPCMore Information
quantum-computing.ibm.com

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

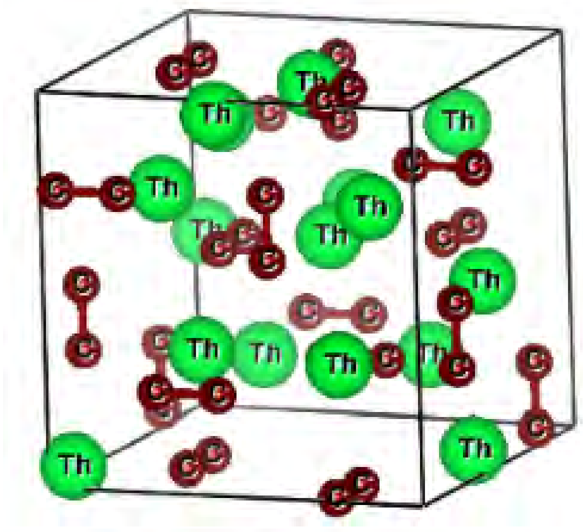
PRACE SoHPCProject ID
2212

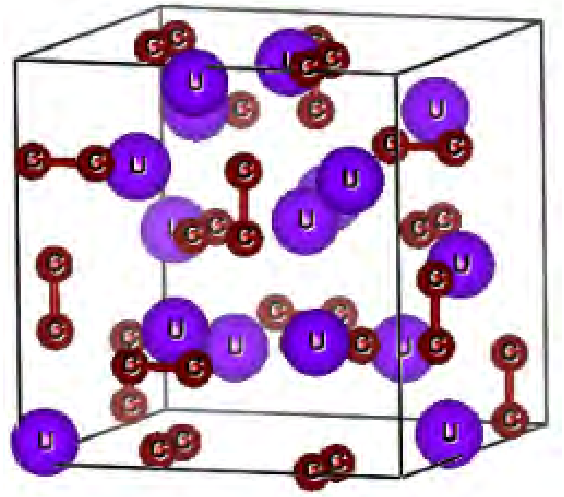
Density Functional Theory (DFT) calculations for Uranium and Thorium sesquicarbide using IT4I supercomputing center

First principle calculations for nuclear fission

Mattia Iannetti, Luigi Camerano Spelta Rapini

Nuclear reactions provide a large amount of energy that can be used to face up the growing energy demand around the world. However, novel fuels are needed for faster and efficient transfer of heat/energy in generation IV reactors. DFT and HPC are the perfect tools to investigate materials properties under extreme conditions.

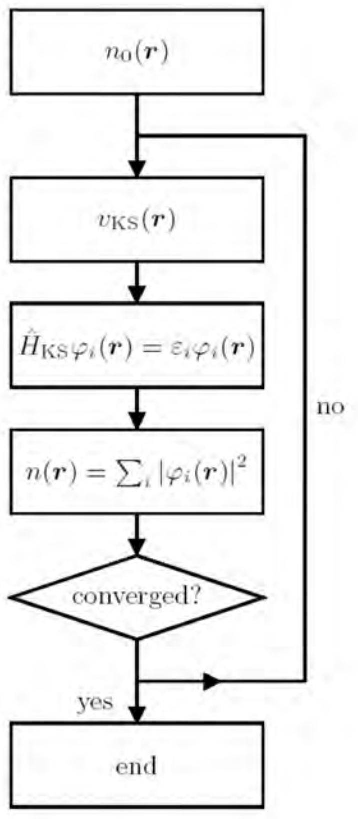




We all experience the rising need of a reliable, cheap and green energy source which can satisfy all the standards of the modern society. Nuclear fission can achieve these goals provided that some improvements are applied.
Density functional theory is the tool we use in the project to investigate electronic structure, phonon dispersion and thermal properties of the materials that could be highly efficient novel nuclear fuels. We was very fascinated when for the first time we played with DFT code to understand the electronic properties of **Molybdenum** and **Palladium**: given the crystal structure and number of valence electrons in a few second one obtains the equilibrium distance among atoms and all the electrons available states which determine the main properties of the materials, isn't it amazing? With the output of a DFT code one can predict exactly the result of an experiment or one can say if your material is a metal or an insulator, it is opaque or transparent without getting your hands dirty.
Furthermore you can say if your material is a suitable fuel for nuclear fission or not, and this is the main issue of this project. The idea is to investigate **Thorium** and **Uranium** carbide and to analyze how the presence of Carbon atoms affects the thermal conductivity and so the efficiency of the fuel.
In particular we want to understand how the heat transfer depends on the phonon dispersion, the heat capacity of single mode, and the relaxation time.
To do so we first have to understand how electrons behaves in these compounds.
Our project uses VASP[1] (Vienna Ab initio Simulation Package) and Phonopy[2] to implement lattice vibration.
This work aims to clarify some of the aspects involved in novel fission reactors.

Density Functional Theory

Density Functional Theory (DFT) is a computational quantum mechanical modelling method used to solve the microscopic problem starting from the charge density of our system. Given a trial electron charge density, for example a combination of atomic orbitals, the self consistent algorithm starts to solve a Schrodinger-like equation to evaluate a new charge density as shown in Figure 1. Once the difference between the starting charge density and the calculated one is small enough, the algorithm converges and the total energy of the system is determined.



$n_0(\boldsymbol{r})$
$v_{\rm KS}(\boldsymbol{r})$
$\hat{H}_{\rm KS}\varphi_i(\boldsymbol{r}) = \varepsilon_i\varphi_i(\boldsymbol{r})$
$n(\boldsymbol{r}) = \sum_i |\varphi_i(\boldsymbol{r})|^2$
converged?
no
yes
end

Figure 1: Figure explaining the self consistency algorithm of DFT method.

Methods

Our DFT calculations were carried out using the Vienna Ab initio Simulation Package (VASP)[1] with a particular computational technique: the projector augmented wave scheme (PAW)[9].[10] One of the main problems in DFT is the electron-electron correlation. To deal with it we use the generalized gradient approximation parametrized by Perdew-Burke-Ernzerhof.[11]
To calculate vibrational properties (phonons), we used supercell method of small displacements as implemented in PHONOPY.[2]

First approach to DFT calculations

As mentioned before we first started to become acquainted with VASP codes with simple materials: BBC molybdenum and FCC palladium.
On these two crystals we first calculated the equilibrium parameters such as the lattice parameter and the bulk modulus. Then, knowing the distance among atoms, Density Of States (DOS) and band structure were calculated. These properties describe the electronic energies and the electronic available states. Then, based on elastic constants (calculated by VASP and code AELAS[3]) we determined mechanical stability of these crystals.

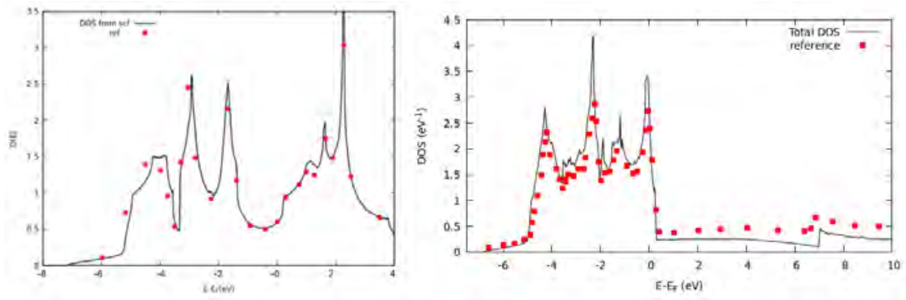


Figure 2: Density of states for Molybdenum and Palladium determined. The DOS is the number of available electron states per unit energy. The red dots are from literature DFT calculations.[457]

Since electrons are fermions (identical electrons cannot stay in the same state for the Pauli exclusion principle), it can be defined an energy (Fermi energy) such that all the states below it are filled while the above ones are empty (at T=0K). The number of states near this energy determine the main properties of the material. For example in the Figure 2 one can see that palladium has more electrons than molybdenum near the Fermi energy.This fact influences the thermal properties related to the electrons and will become important at higher temperatures.

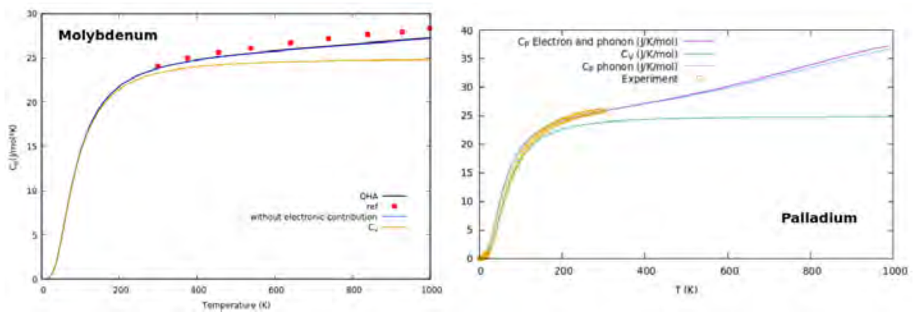


Figure 3: Heat capacity at constant pressure is shown for both Palladium and Molybdenum. The red dots are experimental points.[68]

But wait a moment: to determine the thermal properties we need to consider also the movement of the ions (the so called phonons)! The lattice vibrations that guide the ions movements are determined by so-called Hellman-Feynman forces of all atoms. Then the thermal properties are given by both the electronic and ionic contribution.
It is worth noting that electronic contribution in molybdenum is negligible since the number of the electron near the Fermi energy is very small. Instead for palladium electrons matter! (Figure 3). Another study we did is on the rock salt (NaCl) to understand the phonon behavior for a heteropolar compound. Electronic calculations were done to determine the equilibrium lattice parameter and then atomic vibrations were investigated (Figure 4). Since the NaCl is an heteropolar compound of two atoms, it contains optical as well as three acoustic modes, modes (a,b,c), that goes to zero at Γ (center of Brillouin zone). The transversal optical modes (d,f) have the same energy along certain direction while the longitudinal one (e) has an higher energy. And now you could ask: why are you telling me all of this?
The answer is that we have to do experiments. More specifically, when we want to study a certain compound, we have to be sure that our sample has the same characteristics that we studied on the computer. To do so there are two main techniques: Raman and Infra-Red spectroscopy. These two experiments consist in sampling our material with light. The reflection of the incident beam is strictly related to the motion of the ions in the crystal! That's why we need to know how nuclei moves.

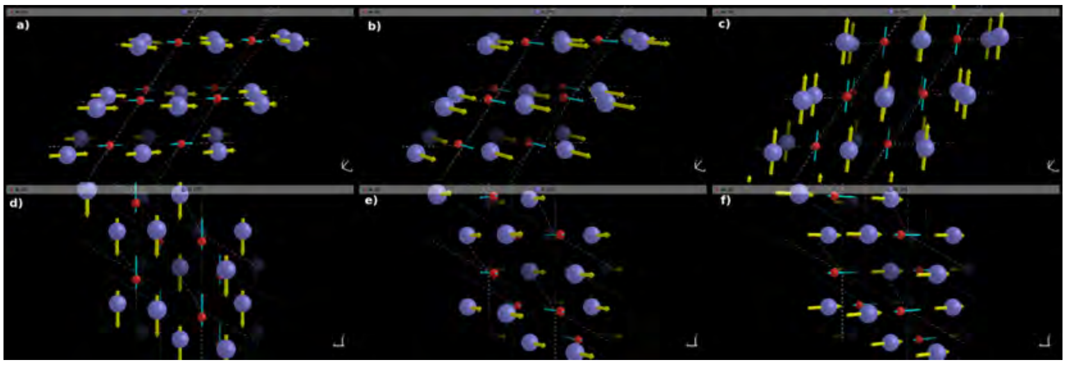


Figure 4: Some examples of displacements of nuclei in $NaCl$ crystal.

Thorium and uranium sesquicarbides

Now that we know how to deal with a new compound we begin looking into these two candidates as novel nuclear fuel, Th_2C_3 and U_2C_3. As for other compounds we need to study the material stability, look for the equilibrium lattice constant and then start calculating electronic properties. But problems never stop! Uranium and Thorium have a very high atomic number(large mass) and so the inner electrons move very fast! Therefore relativistic effects may be not negligible. out the Spin-Orbit Coupling (SOC) for both uranium and thorium sesquicarbide. But wait a minute, what are these bands? Since we are solving a Schrodinger-like equation, we will eventually get quantized energies. But more importantly, just from the fact that our system is periodic, a new quantum number, related with the momenta of electrons, come up. The dependence of the energy on this sort of momenta is called band structure, a dispersion diagram if you like. As you can see, the relativistic correction is visibly not negligible in U_2C_3 since the bands differ a lot, while in Th_2C_3 the SOC is negligible.

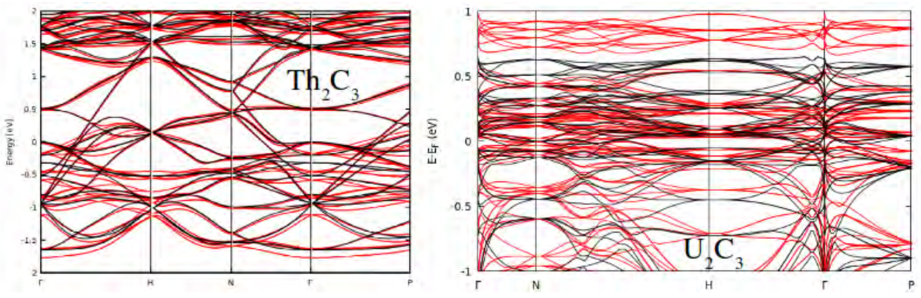


Figure 5: Band structure for both Uranium (up) and Thorium (down) compounds. The black lines correspond to the calculation without SOC the red ones are with SOC

In VASP code there is a second variational implementation of the so called Spin-Orbit Coupling (SOC) that is just the relativistic correction we were talking about. This takes into account the coupling between the famous spin degree of freedom and the orbital moment of the electrons. Without going into the details, let's see some result.

Results and further developments

In this section we report our study on Th_2C_3 and U_2C_3. As said before, once the equilibrium lattice parameter is found, electronic calculations can be done. In the figure 5 it is reported the electronic band structure with and with-

Further investigations should be done on phonon structure and thermal properties of these compounds to see if and how much the presence of carbon atoms affect the heat transport.

References

[1] G. Kresse and J. Furthmüller, Phys. Rev. B 54, 11169 (1996)

[2] "First principles phonon calculations in materials science", Atsushi Togo and Isao Tanaka, Scr. Mater., 108, 1-5 (2015)

[3] Zhang, S. H., and R. F. Zhang. "AELAS: Automatic ELAStic property derivations via high-throughput first-principles computation." Computer Physics Communications 220 (2017): 403-416.

[4] "Formation and electronic properties of palladium hydrides and palladium-rhodium dihydride alloys under pressure" Scientific Report volume 7 Y. Xiao *et al* 14 june 2017.

[5] "Phonon anharmonicity and components of the entropy in palladium and platinum", Physical Review B 93, 214303 (2016) of Y. Shen *et al*.

[6] "Specific-heat study of nanocrystalline palladium" Physical Review B 1october 1995 of Y. Y. Chen *et al*.

[7] Okada, Michio, et al. "Evolution of the electronic structure in Mo1 = xRex alloys." New Journal of Physics 15.9 (2013): 093010.

[8] Chase, M.W., Jr., NIST-JANAF Themochemical Tables, Fourth Edition, J. Phys. Chem. Ref. Data, Monograph 9, 1998, 1-1951.

[9] P. E. Blöchl, Phys. Rev. B 50, 17953 (1994)

[10] G. Kresse and D. Joubert, Phys. Rev. B 59, 1758 (1999)

[11] J. P. Perdew, K. Burke, and M. Ernzerhof, Phys. Rev. Lett. 77, 3865 (1996)

[12] "Thermo physical properties and pressure induced phase transition in thorium and uranium sesquicarbide (Th_2C_3 ,U_2C_3)", J. Appl. Phys. 129, 04 January 2021, of B. D. Sahoo and K. D. Joshi

PRACE SoHPCProject Title

Heat transport in novel nuclear fuels

Luigi Camerano Spelta Rapini

PRACE SoHPCSite

IT4Innovations National Supercomputing Center at VSB – Technical University of Ostrava

PRACE SoHPCAuthors

Luigi Camerano Spelta Rapini and Mattia Iannetti, University of L'Aquila, Italy

Mattia Iannetti

PRACE SoHPCMentor

Dominik Legut, Ostrava, Czech Republic

PRACE SoHPCContact

Luigi, Camerano Spelta Rapini
University of L'Aquila
Phone: +39 388 5897370
E-mail:
luigi.cameranospeltarapini@student.univaq.it
Mattia, Iannetti
University of L'Aquila
Phone: +39 3270127909
E-mail:
mattia.iannetti@student.univaq.it

PRACE SoHPCSoftware applied

VASP, Phonopy, AELAS[3]

PRACE SoHPCMore Information

www.virtouso.org

PRACE SoHPCAcknowledgement

We want to thank Dominik Legut, IT4I supercomputing center and PRACE SoHPC for giving us this opportunity .

PRACE SoHPCProject ID

2213

High Performance Quantum Fields

Apostolos Giannousas
Christopher Kirwan

Lattice QCD can produce experimentally verifiable results at low energies. Observables will be extracted from numerical simulations and software built for the exascale-era will be benchmarked.

Lattice QCD - Introduction

Qunatum Chromodynamics is the ruling theory behind the strong interaction. It is formulated in terms of quarks and gluons, with physical quantities/observables calculated using the Path Integral. Calculating these infinite dimensional integrals with analytical tools is an impossibility. Pertubative techniques do exist, but these fail at the energy scale necessary to describe nuclear interactions.

Lattice QCD provides a non-pertubative approach to these calculations by discretizing our 4-dimensional spacetime such that coordinates take integer values (aka. a lattice). This naturally provides methods to simulate QCD on a computer. A quark field ψ is defined at each point $\psi(n)$, with a gauge link variable U_μ connecting neighbouring lattice points.

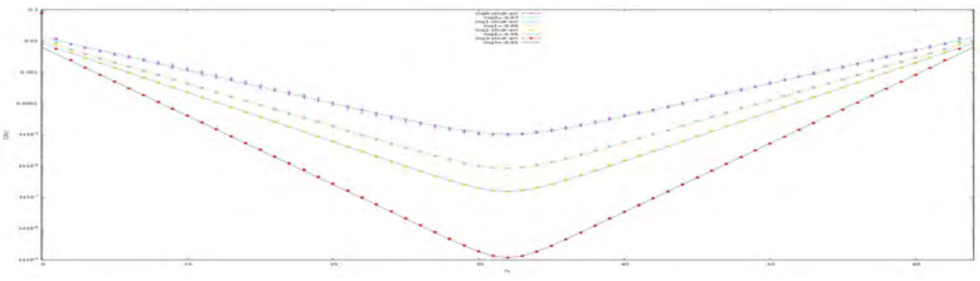
Extracting Hadron Masses with LQCD

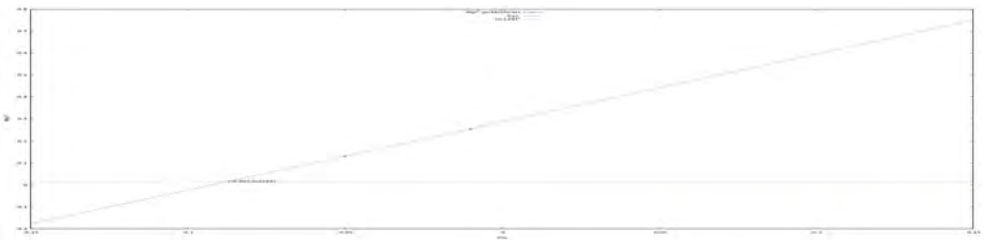
As mentioned earlier, one of the main goals in LQCD problems is the calculation of various quantities. A fairly common task is to perform the so-called hadron spectroscopy calculation. That means determining a hadron mass based on some specific input parameters. One of those input parameters is the gauge field configuration, which directs us to an essential step towards approximating the theoretical solution of the Path Integral. Specifically, to obtain a solution that converges with the Path Integral's solution, we carry out the same experiment with different configurations and take the average output over all of them.

In addition, another parameter that defines the preciseness of the calculation is the input quark mass. This project's motivation was to perform hadron spectroscopy for a pion and find the target quark mass resulting in a pion mass value close to the experimental one. This goal was tackled efficiently by leveraging HPC resources.

Methods

To make the aforementioned LQCD simulations, I utilized a tool called chromaform and tried to optimize it for Hawk and Juwels supercomputers, where I performed all those simulations. The parameters that need to be considered in order to highly parallelize the execution of chromaform are actually the dimensions of the lattice. Since there were fixed lattice dimensions for the following experiments, I made a number of scripts that issued chromaform jobs with those dimensions but used different number of MPI processes and OpenMP threads. Upon finding the optimal configuration for my needs, I proceeded with the essence of the project.


(a) Graph with exponential fitting of the 4 average correlators


(b) Graph with the line representing the relation of the pion mass square and quark mass. The horizontal blue line is at the desired pion mass square

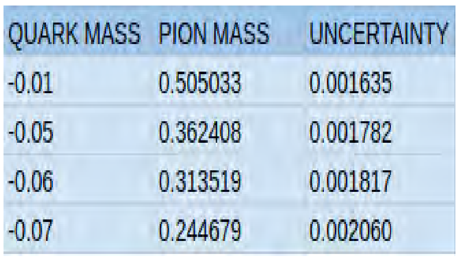
Initially, I did a lot of simulations using the same quark mass of -0.01 but a different gauge configuration each time. Note that the quark mass here is negative due to an additive mass shift introduced during discretization. Afterward, I extracted the pion correlators of those simulations and calculated their average and standard error. Those statistical observables were used to perform exponential fitting and to acquire the pion mass.

The fitting procedure was executed with the help of Gnuplot. However, since Gnuplot does not consider the correlation among configurations, its error estimate can be significantly inaccurate. That happens because the standard error calculation of Gnuplot assumes that the pion correlators are independent variables, which is of course incorrect. In this case, the best practice is to use either the jackknife or the bootstrap method. I opted for the jackknife method as it is simpler to implement. According to it, we should exclude one correlator at a time, calculate the average of the rest and use all those averages to find the final error. That can be viewed as a way to construct pseudo-independent variables(correlators are the variables here), thus making the error calculation more precise.

After completing the above steps, even though we ended up with a pion mass relatively close to the experimental value and a logical error estimate, we wanted to obtain an even better solution. Consequently, I decided to exploit the approximately linear relation between the square of the pion mass and the quark mass. To do this, I repeated the simulations with a quark mass of -0.05 and extracted a new pion mass. Having two pairs of a quark mass and a pion mass square, I created the line they defined, plugged in the desired pion mass square, and extracted the theoretical quark mass I should use to get it.

Results

To begin explaining the results, I should point out that everything is in lattice units which makes the masses dimensionless. The experimental physical pion mass is roughly 135 MeV and translates to a lattice mass of 0.129. By drawing the horizontal line at its square, as shown in figure (b), I found out the input quark mass should be around -0.087. Unfortunately, using such a low mass dramatically increased the difficulty for solutions to converge and forced us to go only as low as -0.07. However, the result was close to the desired one and the prediction about the relation between the two masses appeared to be correct. All these can be observed in the figure below.



QUARK MASS	PION MASS	UNCERTAINTY
-0.01	0.505033	0.001635
-0.05	0.362408	0.001782
-0.06	0.313519	0.001817
-0.07	0.244679	0.002060

Figure 2: Table with results extracted from experiments with 4 different quark masses

Conclusion

The experimentation on hadron spectroscopy with LQCD on HPC resources yielded promising results. Although I could not find a way to try an even smaller mass due to time limitations, the final pion mass was satisfactory. A future PRACE SoHPC participant could alter various other aspects of the chromaform tool and manage to extract even more accurate solutions.

For instance, he could compromise with a smaller requested precision or number of max iterations of the linear system solvers. While that would decrease the accuracy of the final result, it could also significantly lower the complexity of the system's solution. Last but not least, chromaform contains a lot of different solvers, that should be examined one by one since in this scenario one of them could perform better than the others.

Benchmarking with Kokkos

Modern high performance computers have diverse and heterogeneous architectures. For applications to scale and perform well on these modern architectures, they must be re-designed with thread scalability and performance portability as a priority. Orchestrating data structures and execution patterns between these diverse architectures is a difficult problem.

The Kokkos C++ template library is designed to assist the programmer in creating highly portable code, by interacting with shared-memory APIs (such as OpenMP, CUDA, pthreads, etc..) in a unified manner.

The primary aim of this part of the project was to develop a series of benchmarks, primarily reporting memory bandwidth utilization. These benchmarks then target multiple shared-memory backends across the resources available at the Jülich Supercomputing Centre (JSC). To showcase an application to Lattice QCD, a kernel for the Staggered Dirac Operator was developed.

Background

The most demanding computational task in a Lattice QCD simulation is the calculation of the quark propagator. This involves solving a very large system of equations

$$D(U)\psi = \chi$$

where ψ is called the quark propagator, χ is the source term and $D(U)$ is the lattice discretized Dirac Operator. This is generally an incredibly large (approximately $10^8 \times 10^8$ for the latest simulations), sparse matrix.

Simulating staggered fermions involves a staggered transformation of the underlying quark fields. This allows for the Dirac indices on the 12 component spinor $\psi(n)$ to be dropped, and therefore have a 3 component vector $\psi(n)$ at each lattice point. Consequently, the Dirac Operator acting locally on each vector has the convenient form

$$(D\psi)(x) = \sum_{\mu=0}^{3} U_\mu(x)\psi(x+\hat{\mu}) - U_\mu^\dagger(x-\hat{\mu})\psi(x-\hat{\mu})$$

where $\hat{\mu}$ is the unit vector associated with the lattice direction μ. This means that the operator is composed of many small matrix-vector multiplications with the 8 nearest-neighbours on a lattice site n.

Method

The Dirac operator kernel was implemented using the Kokkos library. Multiple different targets were made for both the Juwels-Booster and Jureca supercomputer clusters at the Jülich Supercomputing Centre (JSC). These included compiling with `clang` and `nvcc` (for nVidia GPUs). Various optimizations were turned on at compile time for each specific system arthitecture.

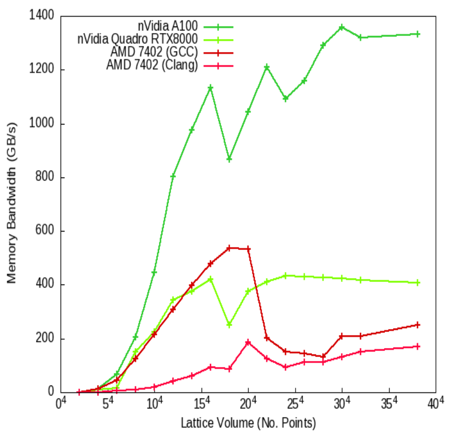
Originally, the quark and gauge fields were represented using single dimensional `Kokkos::Views` and therefore I used a 1-D parallel threading policy. However, in using an 4-dimensional threading policy, it was found that it allowed for cache blocking. Thus, the `Kokkos::Views` that represented the quarks and gauge fields were themselves 4-dimensional, which also made indexing and retrieving `Kokkos::subviews` much more straightforward.

The dimension of the lattice was then varied to measure how memory bandwidth utilization changed as a function of the number of lattice sites.

Adding distributed memory communication with MPI was a much simpler process than anticipated. As Kokkos is a shared memory library, it compliments MPI quite nicely. Internode communication with the CPUs and GPUs (along with intranode communication for the GPUs) was done using the halo exchange method. However, since `Kokkos::subviews` are guaranteed to be contiguous, `Kokkos::deep_copy` was used to copy into and out of `recv` and `send` buffers, without having to implement custom `MPI_Datatypes`. Benchmarks were performed by varying both the problem size and number of MPI processes.

Results

The results of the benchmarks are included in this section. This plot shows memory bandwidth utilisation in GB/s, for shared memory processes on a single node for the two clusters at the JSC, versus lattice volume.



From the above graph, it is clear that as the lattice volume increases, the total memory bandwidth increases, especially for the nVidia A100 GPU. For the CPUs, a drop in memory bandwidth utilisation is seen. This indicates that the memory channels have become saturated. To alleviate this issue, wider registers could be used to sustain higher bandwidth utilisation for larger lattices.

Conclusion

With small attempts at optimizing the code base, but avoiding the inclusion of Kokkos library extensions (such as using Kokkos-SIMD instructions, or Kokkos dense linear algebra kernels), it was found that Kokkos provides a solid foundation for performance portable code bases, especially for memory bound problems such as Lattice QCD. The Staggered Dirac kernels scale well to larger lattices

The ability to produce targets for a wide variety of architectures is certainly an attractive feature for anyone developing code for the newest architectures.

PRACE SoHPCProject Title
High Performance Quantum Fields

PRACE SoHPCSite
JSC (Jülich Supercomputing Centre), Germany

PRACE SoHPCAuthors
apogiannousas@gmail.com, UTH, Volos, Greece
kirwanch@tcd.ie, TCD, Ireland

Apostolos Giannousas

Christopher Kirwan

PRACE SoHPCMentors
Eric Gregory, JSC, Germany
Stefan Krieg, JSC, Germany

PRACE SoHPCContact
Apostolos Giannousas, University of Thessaly
Phone: +30 6957087328
E-mail: apogiannousas@gmail.com
Christopher Kirwan, Trinity College Dublin Phone: +353 86 050 6499
E-mail: kirwanch@tcd.ie

PRACE SoHPCSoftware applied
Chromaform
Kokkos

PRACE SoHPCAcknowledgement
We would like to thank our mentors for their invaluable help throughout the summer as well as the coordinators for making the PRACE SoHPC experience possible.

PRACE SoHPCProject ID
2214

Adapting the Fast Multipole Method to distributed memory

Chitchat, Gossip & Chatter

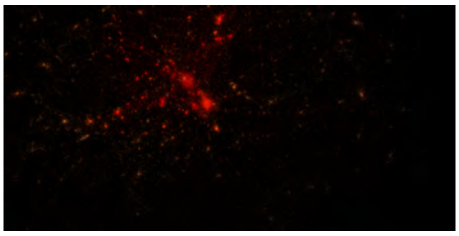
Ignacio Encinas Rubio

Today's supercomputers consist of up to several thousands of computing nodes, and adapting sequential programs in order to leverage their computing power is no easy feat.

N-body simulations are fundamental in many areas of research such as cosmology or chemistry. Performing increasingly big simulations with a very high number of particles forces us to leverage the whole power of today's supercomputing centers. This implies moving from shared memory to distributed memory, and the difficulty of doing so depends vastly on the workload. In this project we have taken the first steps towards having a scalable Fast Multipole implementation. We focused on near field scalability for light workloads as the main use case is MD simulations dominated by long range interactions.

Fast Multipole Method

The Fast Multipole Method is an algorithm developed to speedup N-body simulations. Its importance arises from the fact that it reduces the computational complexity for N-body simulations from $O(n^2)$ to $O(n)$. This is crucial because it allows us to perform bigger and bigger simulations that simply would not be possible with an $O(n^2)$ algorithm even with the best possible resources. For this achievement, the Fast Multipole Method is regarded as one of the top ten algorithms of the 20th century.

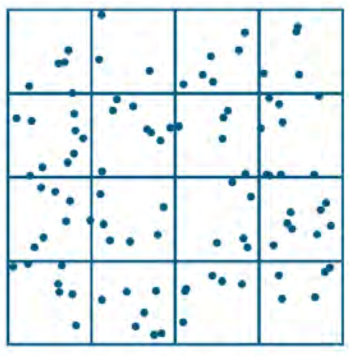


A natural question is to wonder how this complexity reduction is achieved with respect to the naive solution, so we will give a brief explanation on how it works.

Some interactions such as the Coulomb and Gravitational forces decrease in intensity proportional to $\frac{1}{r^2}$ where r represents the distance between the two interacting particles. As r increases, the force becomes more and more negligible and we're tempted to just ignore it. It would be great if we could just ignore far particles and speedup the computation that way, but the problem is that while doing so we would be accumulating an unbounded error over time, making our efforts totally worthless.

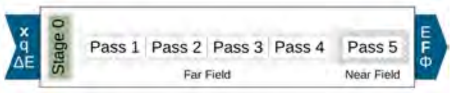
The cleverness of FMM comes from approximating the interaction of far particles, while making the pairwise computation for close particles. This approach bounds the total error and produces usable results.

The first step is to divide space up to depth d as illustrated in the figure. Then, we compute an approximation for each box that will represent every particle in it.



d = 2

Once we have set up everything, we can proceed with the actual calculations. These are split in different passes that perform very specific tasks. Pass 1 to Pass 4 work with the representation of boxes while Pass 5 works with individual particles for the near field interactions.



Too many messages

As we have seen, these tree-like data structures play a fundamental role in the Fast Multipole Method. This is relatively simple when dealing with shared memory but if we want to move away to distributed memory problems arise.

Distributing ownership of every box on every level in a robust way that doesn't completely fail when the simulation isn't homogeneous is very complicated. Furthermore, in the far field passes we have to access the tree data structures very extensively, and this means that we have to perform lots of communications.

For these reasons we decided to focus on the near field pass as its adaptation to distributed memory does not suffer of the problems mentioned above, and allows for very simple implementations that yield correct results from which we can improve gradually.

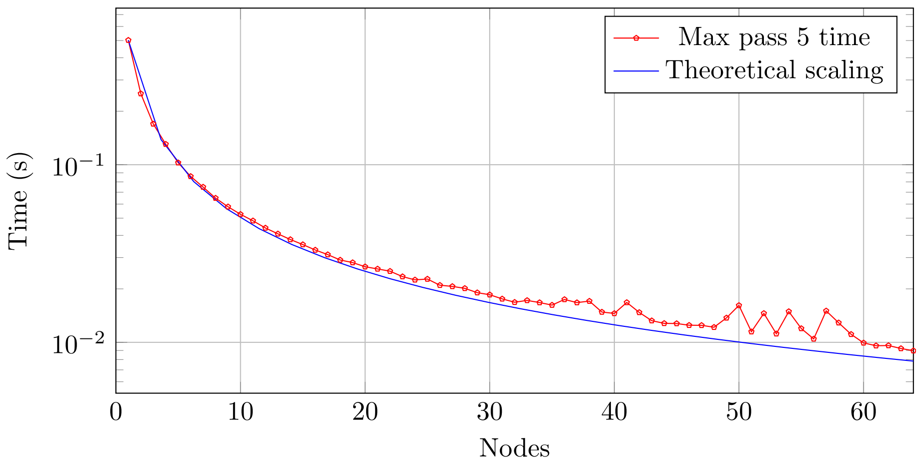
Near field

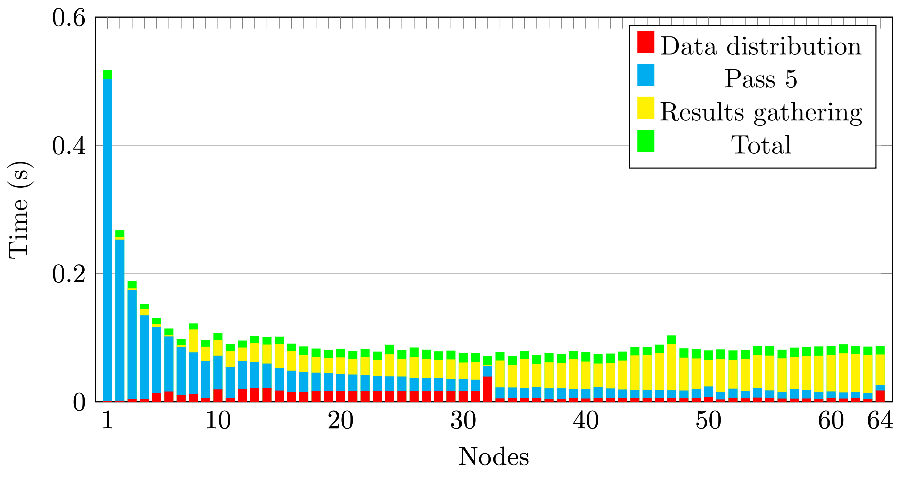
Our initial implementation for the near field consisted on communicating every bit of information from our principal node and let the code work and improve from there. We expected that the initial distribution would be the main bottleneck and we discussed some complex schemas to mitigate it, but the results were surprising. The major slowdown when scaling up in the number of cores actually was in the gathering of results on the principal node. Our main hypothesis were three:

- The message size is too big
- The naive MPI_Reduce implementation chained up delays summing up to a significant amount
- Load imbalance presents itself when scaling up in the number of processors

Results

We measured the maximum time that the actual computation took in every node in order to search for load imbalance. Results showed that this could not explain the results gathering slowdown as the workload distribution was highly homogeneous. On the other hand, we added the necessary bookkeeping that allowed us to retrieve the minimum amount of information from each node.





With these modifications we were able to evaluate all three hypothesis as we succesfully reduced the message size, removed the naive MPI_Reduce and discarded the possible load imbalance.

The results were obtained running a modified FMM implementation that only performed Pass 5 as we just wanted to study the performance on the data distribution and result gathering.

It should be taken into account that the communication latencies can be hidden in the complete version by performing the far field passes while waiting for communication to finish.

Further work is required in order to understand this unexpected bottleneck and how to circumvent it.

Conclusion

Results show that our goal is not as easy as one might think at first sight. We expected major resistance towards scalability but were surprised by its fiercity. We were not able to perform any major breakthroughs.

On the bright side, the naive data distribution performed surprisingly well, and combining this with the fact that the gathering latencies can be hidden by the calculations performed on the far field passes means that we can get some performance with a low number of nodes.

PRACE SoHPCProject Title
Chitchat, Gossip & Chatter,- How to efficiently deal with communication

PRACE SoHPCSite
Jülich Supercomputing Centre, Germany

PRACE SoHPCAuthors
Ignacio Encinas Rubio, Spain

PRACE SoHPCMentor
Ivo Kabadshow, DE, Germany

PRACE SoHPCContact
Ignacio Encinas Rubio
E-mail:
ignacioencinasrubio@gmail.com

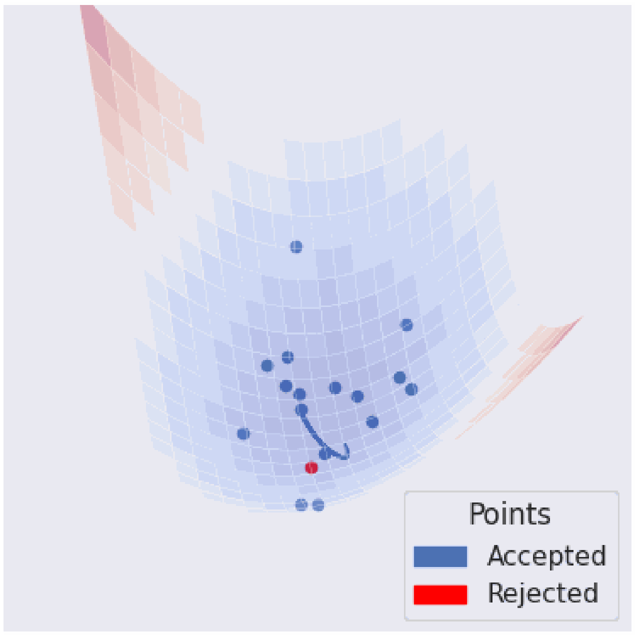
PRACE SoHPCSoftware applied
C++, MPI

PRACE SoHPCAcknowledgement
Images authored by Michael L. Umbricht, Ivo Kabadshow, David Haensel and Andreass Beckmann

PRACE SoHPCProject ID
2215

Ignacio Encinas Rubio

Multi-GPU Hamiltonian Monte Carlo



HMC is widely used in probabilistic programming, as part of fitting many-parameter models. Our implementation - created from scratch - is scalable, portable & based on Physics.

Generating samples from a given probability distribution function is a surprisingly difficult task, and Hamiltonian Monte Carlo (HMC) solves this challenge in an elegant way. HMC is widely used in probabilistic programming packages, such as STAN[1] and numpyro,[2] which use the algorithm to sample parameters from posterior probability distributions, which govern the distribution of model parameters given some data. The size and number of parameters of some statistical models, such as those in epidemiology, justifies parallelized HMC schemes.

There were two principal goals for this project. First, we had to implement from scratch in Python the HMC algorithm as presented and discussed in previous works.[3,4] For such a purpose, we opted for a physics based approach, as this allowed for a more intuitive interpretation and future extension of the method. We assumed we were given a system with particles moving in a multidimensional space and these do not interact with each other. In addition, each particle had its own mass and the system could be heated up and cooled down by considering the temperature as a parameter.

Secondly, we adapted our implementation with the goal of it being able to run on multi-CPU or GPU clusters by simply changing one line of code.

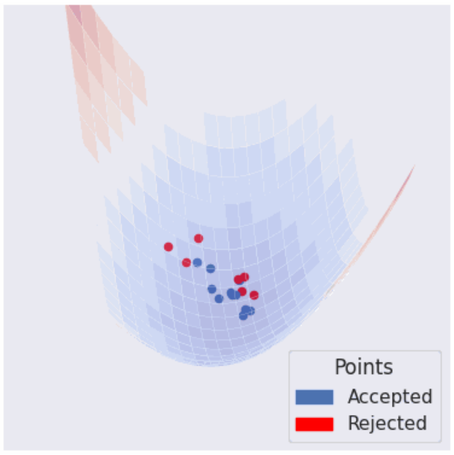


Figure 1: Proposals generated by the Metropolis-Hastings algorithm. Note the high rejection rate and high correlation, even with 2 dimensions.

Theory

To begin with, let us assume that we have a model, and some data which is supposed to be explained by the model. Through Bayesian inference the probability distribution of the model parameters can be deduced. Generating parameter samples from this distribution is difficult or impossible via inverse transform sampling, but can be done via HMC.

Bayes Rule

We recall that Bayes theorem states that given the *prior* probability $p(\mathbf{q})$ and a set of observed data y whose *likelihood* is $p(y|\mathbf{q})$, then the *posterior* probability $p(\mathbf{q}|y)$ is

$$p(\mathbf{q}|y) = \frac{p(\mathbf{q})p(y|\mathbf{q})}{p(y)},$$

where $p(y)$ is a normalizing constant, which may be ignored. Typically, $p(\mathbf{q}|y)$ corresponds to the distribution we want to sample from, which we will call $\pi(q)$ from now on. HMC samples from this distribution for the parameters, as outlined below. This is done to calculate expectation values for the parameters of the distribution, along with standard deviations.

Markov Chain Sampling Methods

Let us assume we are interested in generating samples from a probability distribution function or density function $\pi(\mathbf{q})$ with parameter $\mathbf{q}$ on a D-dimensional space, Q. Furthermore, $\pi(\mathbf{q})$ is not easily invertible or is not available in closed form.

The subset $T \subset Q$ where the product $\pi(\mathbf{q})d\mathbf{q}$ such that $\mathbf{q} \in T$ is non-negligible is called the typical set. This region, called the typical set, is where we want to sample from. Notably the volume of T relative to the region surrounding T decreases as D increases.

There exist many sampling techniques to generate points from $\pi(\mathbf{q})$ and one of the most widely used techniques is a class of statistical methods called Markov Chain Monte Carlo (MCMC). These methods, intuitively, consist of randomly navigating through the space spanned by parameter $\mathbf{q}$, commonly called *parameter space*.

Most MCMC methods work by generating the proposals for the next sample ($\mathbf{q}_{n+1}$) in proximity to the current sample ($\mathbf{q}_n$). For instance, the Guassian Metropolis-Hastings method involves taking a random Gaussian step in parameter space to get the next proposal, which we denote by $\mathbf{q}'$. The proposal is then accepted or rejected with probability $\pi(\mathbf{q}')/\pi(\mathbf{q}_n)$. The rejection of samples at a lower probability density causes the walk to “stay on track”, and for computational resources to be focused on the typical set, where the density of the target distribution is significant.

This simple Metropolis-Hastings method, shown in Figure 1, struggles with high-dimensional distributions, when D is large, as the volume surrounding T is much larger than the volume of T itself, meaning most proposals are in regions of Q where $\pi(\mathbf{q})d\mathbf{q}$ is negligible and are, therefore, rejected. This is not efficient at best and totally infeasible if D is sufficiently high.

This is where HMC steps in. Work on the random motion of smoke particles in air (brownian motion) by Einstein and others established that the motion of particles in a gas can be modeled using Markov chains. Inverting this, some markov chains can be considered as particles. Suppose that we are given a particle whose position vector $\mathbf{q}$ is on a D-dimensional space; keep in mind that $\mathbf{q}$ is the parameter we are interested in. HMC doubles the parameter space Q by adding an extra parameter, which we refer to as *momentum*, $\mathbf{p}$. Then, we have a joint probability distribution function depending on $\mathbf{q}$ and $\mathbf{p}$, that is, $\pi(\mathbf{q},\mathbf{p}) = \pi(\mathbf{p}|\mathbf{q})\pi(\mathbf{q})$. Notably, in physics we often sample from the *canonical distribution*, which is given by $\pi(\mathbf{q},\mathbf{p}) = \exp(-H(\mathbf{q},\mathbf{p}))$, where $H(\mathbf{q},\mathbf{p})$ is the Hamiltonian or energy value at $(\mathbf{q},\mathbf{p})$.

We notice that $H(\mathbf{q},\mathbf{p})$ can be written as the sum of kinetic $K(\mathbf{p},\mathbf{q})$ and potential energy $V(\mathbf{q})$ as follows: $H(\mathbf{q},\mathbf{p}) = -\log\pi(\mathbf{q},\mathbf{p}) = -\log\pi(\mathbf{p}|\mathbf{q}) - \log\pi(\mathbf{q}) = K(\mathbf{p},\mathbf{q}) + V(\mathbf{q})$. This means that each point in the parameter space is assigned a potential energy $V(\mathbf{q}) = -\ln\pi(\mathbf{q})$. To illustrate this, imagine a 2D Gaussian distribution $\pi(\mathbf{q}) = \exp(-q_1^2 - q_2^2)$ (lacking normalization). This has a bowl shaped parabolic potential energy function $V(\mathbf{q}) = q_1^2 + q_2^2$. We set $K(\mathbf{p},\mathbf{q}) = (\mathbf{p}^T \cdot \mathbf{p})/2m$. Such a choice of $K(\mathbf{p},\mathbf{q})$ is usually implemented as discussed in.[3,4] Besides, we state that the Hamiltonian captures the "geometrical information" of T and that $K(\mathbf{p},\mathbf{q})$ is non-unique.

To go from one sample to the next, we give the “particle”, whose mass is m, a random momentum $\mathbf{p}$ from a Maxwell-Boltzmann distribution, which approximates the motion of particles in an ideal gas. In practice this means momentum is drawn from a multivariate normal distribution with a mean of zero and the standard deviation for each coordinate as given below.

$$\sigma_{p_i}^2 = \sigma_{p_{i+1}}^2 = ... = mk_bT,$$

for $i = 1, \ldots, D-1$.

We then simulate the evolution of the system for a certain number of timesteps, which requires solving Hamilton’s equations for $\mathbf{p}, \mathbf{q}$. This is done using a sympletic integrator - which conserves the Hamiltonian, i.e. energy - such as the leapfrog method. We recall that the force F acting upon the particle is the negative of the gradient of the potential with respect to position; that is

$$F = -\nabla V(\mathbf{q}).$$

To concretize the evolution, imagine a hockey puck being given a random momentum in a large, parabolic bowl. In an ideal world, this method of generating proposals negates the need for an acceptance/rejection step, for reasons outlined in the references.[3,4] However, in reality, a small number of proposals are rejected due to numerical integration errors. Nonetheless, with a suitable time-step most proposals are accepted and samples have a lower correlation than those produced by Metropolis-Hastings.

Methods

The implementation was made using Google’s jax, which is NumPy accelerated on GPU. Jax features just-in-time compilation, automatic differentiation and parallelization of evaluation by means of vectorization via ’vmap.’ MPI was used to run the program on multiple GPUs, where vmap was used to run multiple particles in parallel as illustrated in Figure 2.

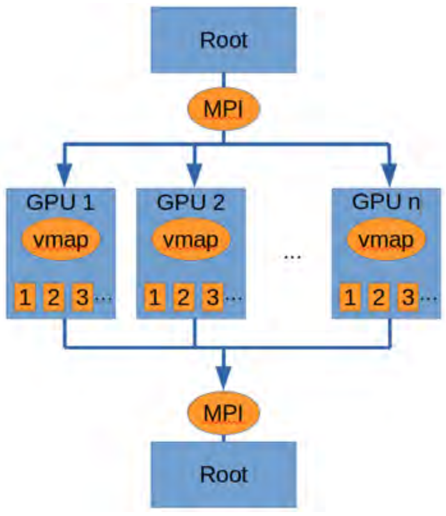


Figure 2: Parallel architecture.

Numpyro played an important role in the project, as it was used to set up the probabilistic models and calculate the log posterior distribution from observations. The posterior’s were then sampled from using our HMC kernel.

Results

We achieved the initial goal of fitting probabilistic models on multiple GPUs. The implementation is also extremely portable, as it can be run on CPUs or GPUs easily, with no requirement for all GPUs to be of the same architecture. Figure 3 shows the scaling of the implementation with the number of particles on a single CPU, a single GPU and two GPUs of different architectures.

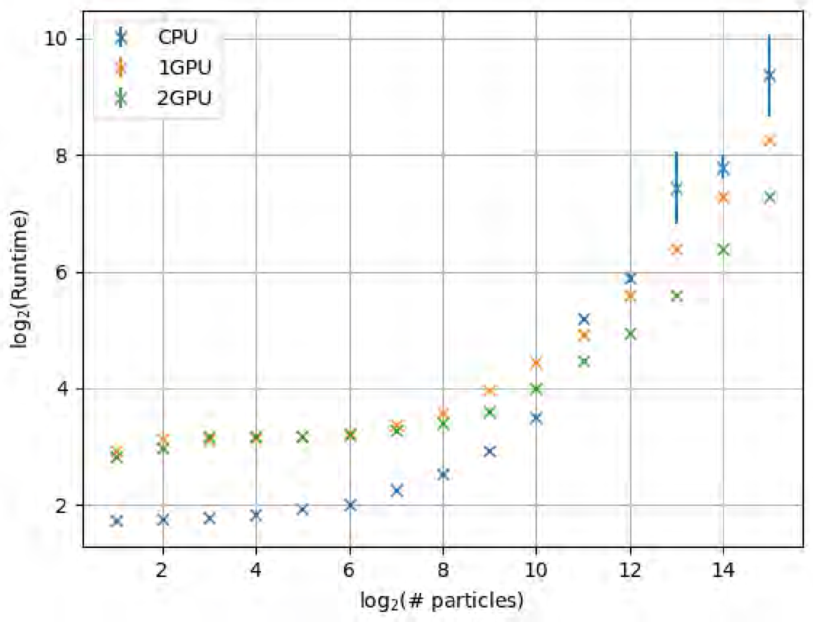


Figure 3: A GTX 1080Ti and RTX 3050 were used, inferring the bias of two coins.

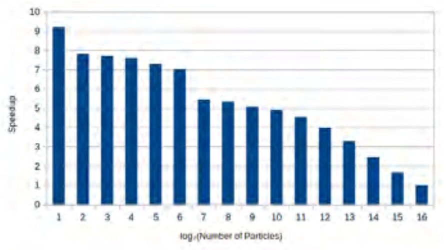


Figure 5: Speedup vs Number of Particles on one V100 GPU.

The speedup of the algorithm with the number of V100 GPUs used is plotted in Figure 4, whilst the change in run time on one GPU as the number of particles increases is shown in Figure 5. The weak scaling in both plots is a consequence of the model being far too small to fully utilize the GPUs, meaning much of the run time is overhead. We limited ourselves to small models which can be solved analytically because this allowed us to verify the correctness of the results obtained with our implementation.

Discussion/Conclusions

Overall we are pleased with our progress and the results so far have been promising. In the future, we plan to fit some larger models and explore how tuning physical parameters can affect our performance.

One promising avenue is simulated annealing. Let us say we have a bi-modal distribution function (ie a pdf with two local maxima). To start we set momentum's with a high temperature, meaning the entire potential landscape (and hopefully each mode) is more readily explored. We gradually lower the temperature before generating samples from the chains, so the tails of the distribution are not over represented.

Also, it is worth noting that current results have been obtained with code that has not yet been optimized. Profiling reveals significant host-device communication which, in combination with the small models used, explains the limited scalability observed so far. Eliminating these transfers and increasing the model size in the coming weeks should lead to major improvements in scalability.

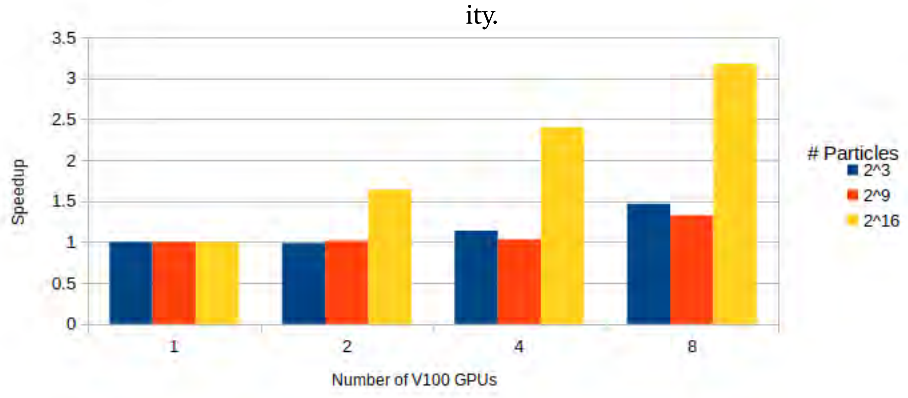


Figure 4: Speedup vs number of V100 GPUs for various numbers of paricles.

References

[1] Stan Reference Manual, Version 2.30. Chapter 15.

[2] Numpyro documentation.

[3] Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. arXiv preprint arXiv:1701.02434.

[4] Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res., 15(1), 1593-1623.

PRACE SoHPCProject Title
Multi-GPU Physics Based Hamiltonian Monte Carlo

PRACE SoHPCSite
Hartree Centre - STFC, United Kingdom

PRACE SoHPCAuthors
Bruno Rodriguez Carrillo, Mexico
Thomas Warford, UK

Bruno Rodriguez

Thomas Warford

PRACE SoHPCMentor
Anton Lebedev, Hartree Centre - STFC, UK

PRACE SoHPCContact
Thomas Warford, University of Manchester
E-mail: thomas.warford@student.manchester.ac.uk
Bruno Rodriguez Carrillo, EPFL
E-mail: bruno.rodriguezcarrillo@epfl.ch

PRACE SoHPCSoftware applied
jax, mpi4jax, numpyro

PRACE SoHPCMore Information
github.com/google/jax,
github.com/mpi4jax/mpi4jax,
github.com/pyro-ppl/numpyro

PRACE SoHPCAcknowledgement
Thanks to Anton Lebedev for mentorship and contribution to code. Thanks to Hartree Centre and PRACE.

PRACE SoHPCProject ID
2216

HPC System Analytics

Yağmur Akarken

What would happen if you knew what was going on on the node you are running your job on? In the project, it is aimed to perform anomaly detection by analyzing operational logs with machine learning methods.

High performance analytics project is a continuation of the previous work, A Holistic Analysis of Datacenter Operations. In the previous study, a comprehensive report was prepared on subjects such as resource usage, energy, and workload characterization by examining pre- and post-Covid operational logs. It is demonstrated that the benefits of holistic analysis method by applying it to the operations of a datacenter infrastructure with over 300 nodes. Also, it is suggested that such observations can help immediately with performance engineering tasks such as predicting future datacenter load, and long-term with the design of datacenter infrastructure. Therefore, it is proposed to analyze operational logs with machine learning methods. Starting from the previous work where operational logs of the cluster computer LISA are characterized, the project proposes a continuation that would bring in scope additional data sources, like Prometheus. It is planned to evaluate an anomaly detection method on this newly created dataset.

With this project, it is desired to produce a dataset by using the dataset created for the previous study and expanding it further. The dataset will contain information from different sources. These resources are XALT, Prometheus, SLURM and EAR. While EAR provides information about energy usage, SLURM provides information flow at job level. XALT collects job level information about the libraries and executables that user's access during their jobs and Prometheus keep the information at the node level. Our aim is to pre-process the data we obtained with Prometheus, train it with a machine learning method and perform anomaly detection.

A Close Look at the Data

The data we originally used comes from 280 CPU nodes in the Lisa cluster. A datapoint is generated every thirty seconds and we can access this datapoint through Prometheus' API. Each datapoint has 80 different features. These features provide information about the node, from CPU usage to network usage.

There is a certain amount of normalization on the accumulated data. For example, all nodes selected as data source are CPU nodes, data accumulated on the same days and at the same times are used for nodes that are desired to be used in the dataset. But some features are not uniform.

For example, the read pages total feature of the data point you see in the figure uses fs17 and fs22 file system files. But another node can use fs21 and fs18. There are multiple features like this. We used some aggregation methods to make these features uniform. Apart from that, the Surfsara power usage feature seen in the same figure carries information about the CPU usage of the node. In the studies conducted with the machine learning method, the power usage feature was primarily used as it would be easier to predict because it follows a certain trend. For example, if the CPU usage is high on a node at one data point, it will likely be high at the next data point.

Understanding Data

Normalized data continued to be processed with principal component analysis. PCA is an unsupervised algorithm that is mainly used for dimensionality reduction but can also be used for visualization, noise filtering or extraction. We wanted to use PCA because we wanted to see if clusters are formed in the data after data visualization. We also wanted to use the data obtained after dimensionality reduction.

We started PCA analysis with a dataset containing more than four million data points. Then we experimented the dataset with PCA's explained variance ratio. In these studies, we tried to find out how many components can cover information at the optimum level. As a result of this study, as you can see in the graphic, we realized that we could cover 97.7% of the information about data by using 15 components, and we continued our next work by using these 15 components.

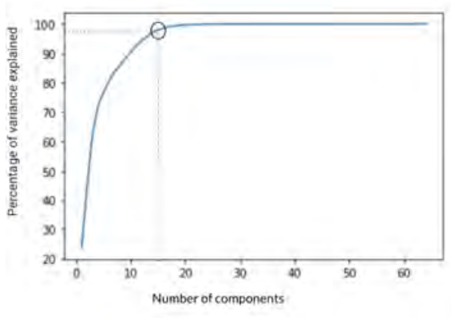


Figure 1: Variance explained by different number of components. After these results all experiments are made by 15 components.

Then we analyzed the features that provide the maximum variance. As a result of this study, we found that disk's written bytes, node core's temperature, node network related features had the most impact.

Outliers with PCA

Afterwards, we took a new step to create our new dataset. We found some outliers that we could detect using the components we obtained because of PCA, and we cleaned our dataset. While detecting the outliers, we used the reverse transforms of the principal components, which we obtained from PCA, and measured the mean square error. As a result of this measurement, we got the results you see in the figures. When we look at the distribution, most data points have an error rate in the range of 0-5, and then these numbers are gradually decreasing. With these results, we continued to work with data points in the range of 0-5 error rate. This corresponds to 95% of the dataset we have.

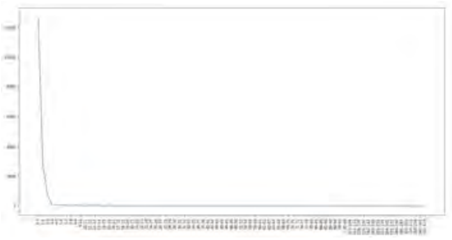


Figure 2: MSE distribution for all datapoints

Visualization of PCA

In order to provide visualization on the data, we reduced the number of components to 3 and performed PCA analysis again. The image we obtained as a result of this analysis looks like a line in a 3-dimensional plane. What we hoped to see here was actually the formation of some clusters and continuing to work by selecting some features from there. While investigating why we got this linear result, we realized that we were getting similar results with small subsets of the dataset, and we started checking the eigenvalues. In this part, we used the Levene's test. But the Levene's test assumes independent observations, in a temporal setting, but the data we are working on is correlated. As a result, we did not get very successful observations and we continued to use all the features we had.

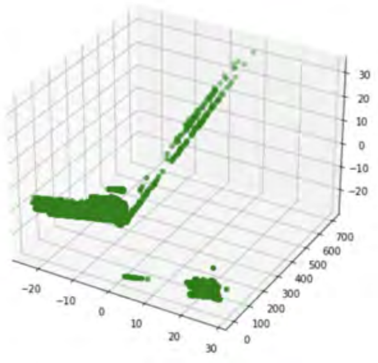


Figure 3: Linear PCA result by using three components

Machine Learning Method for Anomaly Detection: LSTM

After obtaining the dataset, we wanted to choose a machine learning method and train for anomaly detection using the power usage feature. LSTM is a deep learning architecture consisting of a cell structure with input, output and forget gates. While the input gate manages the fed input, the output gate manages the result of the cell. The forget gate helps the cell to be used as memory and decides what is and will not be remembered. Thanks to these structures, LSTM is a technique used for time-series data, with which we can access long-term observations.

In LSTM, we first estimated the CPU usage as it follows some certain trend and easy to predict. We decided that we could do an anomaly detection with a model that could make accurate predictions for CPU usage for the next steps.

Basically, what we did was use the next timestamp data as the predictor for the current input and visualize the test data to see if LSTM could learn something. The mean square error we handled with this random model is 1.29.

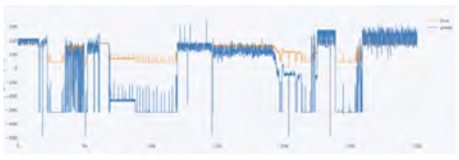


Figure 4: The measurement on test set with random model which makes predictions directly using next datapoint's power usage data

Afterwards, we started our first work with LSTM using information about a single node. Since LSTMs are powerful models, they started to overfit on such a small dataset.

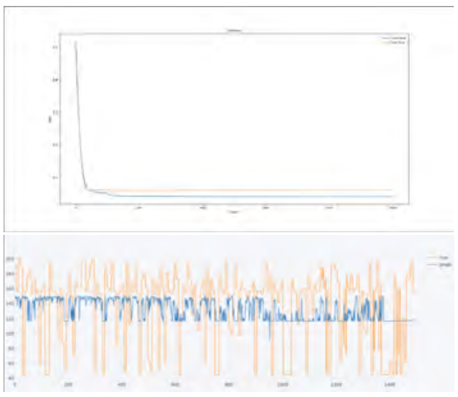


Figure 5: Initial loss curves and predictions belongs to training and testing by using LSTM

We also played with different parameters to overcome this overfitting. These parameters include dataset size, batch size, input sequence length, number of epochs, hidden layer size and output size.

Table 1: Training dataset MSE Loss

seq length \ hidden layer	2	4	8	16	32	64
4	0.054	0.177	0.021	0.018	0.017	0.016
8	0.051	0.030	0.017	0.018	0.016	0.015
16	0.043	0.034	0.031	0.016	0.016	0.016
32	0.049	0.032	0.022	0.018	0.016	**0.014**
64	0.034	0.024	0.022	0.018	0.016	0.016

Table 2: Test dataset MSE Loss

seq length \ hidden layer	2	4	8	16	32	64
4	0.069	0.084	0.025	0.046	0.044	0.042
8	0.092	0.036	0.095	0.035	0.059	0.071
16	0.212	0.058	0.035	0.022	0.029	0.034
32	0.048	0.126	0.068	0.071	0.074	0.038
64	0.037	0.035	0.050	**0.014**	0.021	0.053

Final Results with LSTM

As the final model, we used a model with a hidden layer size of 16, input length of 64, and a batch size of 512. In order to avoid overfitting, we have increased our dataset size ten times, and instead of using all features, we have started using 45 features, which corresponds to 64% of all features.

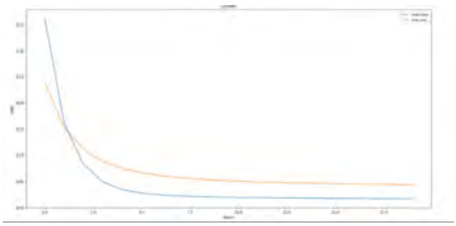


Figure 6: Final LSTM model loss curves for training and testing

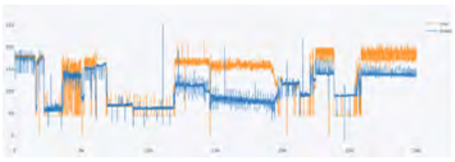


Figure 7: Final LSTM model's predictions and truth values

Conclusion & Discussion

Based on our assessments, LSTM is promising to be good at predicting, especially in areas where data spikes. But as always, there is room for improvement. The next step is to create clusters for normal data points by clustering the outputs on the dense layer of LSTM, and to consider the data points other than this as outliers.

The study can be easily expanded and usage areas can be found. First of all, after clustering is done, predictions with LSTM are planned to be made for all features. Afterwards, it is desired to perform visualization using these predictions in the LISA cluster. In this way, users will be able to have information about the node and closely monitor the nodes they run their jobs on. Another study that could follow this is to do anomaly detection using hierarchical VAE instead of LSTM. Thanks to these kind of studies, we hope to learn more about the use of clusters and supercomputers, and then expand the project at job and system levels and turn it into a useful software for users.

Acknowledgements

Special thanks to my mentors Damian Podareanu, Caspar van Leeuwen, who helped me throughout the whole project, and Carlos Teijeiro Barjas who helped me find accommodation in Amsterdam and answered my questions. In addition, thanks to Duncan Kampert and Bryan Cardenas Guevara, who patiently answered my questions throughout my work. Also many thanks to everyone on the Surf ML team, it was a great pleasure working with them.

References

[1] Versluis, Laurens and Cetin, Mehmet and Greeven, Caspar and Laursen, Kristian and Podareanu, Damian and Codreanu, Valeriu and Uta, Alexandru and Iosup, Alexandru (2021). A Holistic Analysis of Datacenter Operations: Resource Usage, Energy, and Workload Characterization – Extended Technical Report.

PRACE SoHPCProject Title
HPC System Analytics

PRACE SoHPCSite
Surf, Amsterdam, Netherlands

PRACE SoHPCAuthors
Yağmur Akarken, Koç University, Turkey

PRACE SoHPCMentor
Damian Podareanu, Surf, Netherlands
Caspar van Leeuwen, Surf, Netherlands Carlos Teijeiro Barjas, Surf, Netherlands

PRACE SoHPCContact
Yağmur, Akarken, Koç University
Phone: +90 532 5149592
E-mail: yagmurakarken22@gmail.com

PRACE SoHPCSoftware applied
PyTorch, scikit-learn, multiprocessing

PRACE SoHPCMore Information
https://scikit-learn.org/stable/
https://pytorch.org/

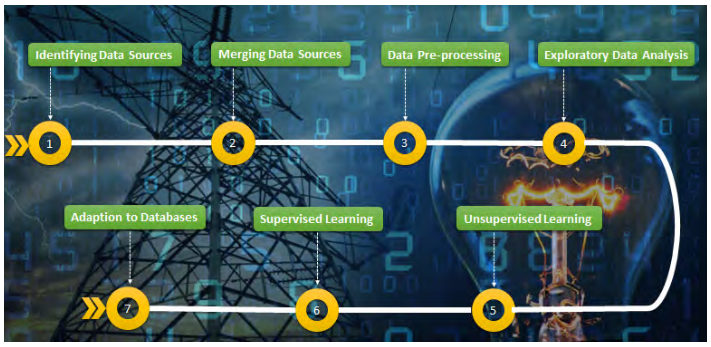
PRACE SoHPCAcknowledgement
Thanks to my mentors Damian Podareanu, Caspar van Leeuwen and Carlos Teijeiro Barjas and also to Surf ML Team.

PRACE SoHPCProject ID
2217

Yağmur Akarken

Big data for electricity

Lucía Absalom Bautista



My projects this summer involve two different goals. On the one hand, learning parallelisation and the use of Rmpi to implement our code on a supercomputer, and on the other hand, the comparative study between ground truth clusters and clusters generated with MSSC.

Two different projects have been occupying my time this summer. Both have made me enjoy and learn, forging my knowledge in the areas of statistics and HPC. In this article I will try to review them both in the same way I have been learning about these topics.

Parallelization and open MPI with R

This was the main goal of the internship and where I spent most of my time doing research. Although I am very familiar with R and artificial intelligence algorithms, this was my first time using parallelization and using a computer with several nodes.

The Data set

The dataset we are working on consists of the recording of electricity consumption for different consumption units collected every 15 minutes. We also have some extra data regarding the date, the type of consumer, the day cathegory (holiday or normal day), etc. All the data is stored on a MongoDB Data Base and stores on a cluster. We retrieve this data using an API getting a json object. A picture of how the data looks??

The Code

The goal of our code is to compute several forecasting models to predict electricity consumption for our dataset. These models are the following:

- `pred_model_GLM_small`: a model usign generalised linear models.
- `pred_model_clust_NN` : a model using Neural Network.
- `pred_model_clust_RF`: a model using Random Forest.

We have had two main tasks concerning the code:

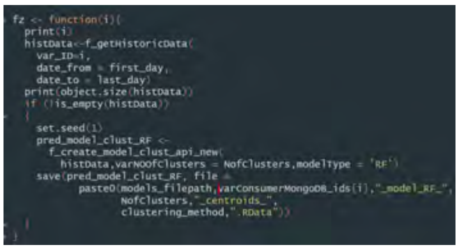
- Parallelization
- MPI

Parallelization

Our main goal in this part is to carry out several processes simultaneously within a node. Firstly I had to clean up the code from small errors related to NA processing and insufficient data for model computation.

After creating an fz function (see Figure 1), which wraps the desired code, we only had to make use of the `doParallel` library, an R library specifically created for parallelisation tasks.

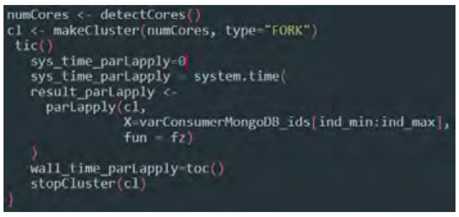
Within it we find several functions such as `%dopar%`, `mcapply`, `parSapply`, `parLapply` that (except for some differences), take as arguments the number of clusters, fz function and the vector with the IDs needed to calculate the models.



```
fz <- function(i){
  print(i)
  histData<-f_getHistoricData(
    var_ID=i,
    date_from = first_day,
    date_to = last_day)
  print(object.size(histData))
  if (!is_empty(histData))
  {
    set.seed(1)
    pred_model_clust_RF <-
      f_create_model_clust_api_new(
        histData,varNOofClusters = NofClusters,modelType = 'RF')
    save(pred_model_clust_RF, file =
           paste0(models_filepath,varConsumerMongoDB_ids[i],"_model_RF_",
                  NofClusters,"_centroids_",
                  clustering_method,".RData"))
  }
}
```

Figure 1: Function fz

In figure 2 we can see an example of the resulting code after the use of the parLapply and system.time function which will allow us to measure the time it takes to run.



```
numCores <- detectCores()
cl <- makeCluster(numCores, type="FORK")
 tic()
   sys_time_parLapply=0
   sys_time_parLapply = system.time(
   result_parLapply <-
     parLapply(cl,
               X=varConsumerMongoDB_ids[ind_min:ind_max],
               fun = fz)
   )
   wall_time_parLapply=toc()
   stopCluster(cl)
}
```

Figure 2: Paralellization code

So this code is running perfectly within one node, the problem is that when we try to run more than, let's say 50 ID's, it's start to get computationally more expensive and consequently much slower to generate (for 100 ID's the code was taking aorund 3 horus to run on login node). So this is where MPI comes in.

MPI

Here we work with two different files, a Master file and a Slave file. In the master file we will spawn the slaves and they will do the computations, each one of them for a certain number of ID's, thus distributing the calculations. When finishing, the slaves will return the information to the master. We can see this work flow on Figure 3.

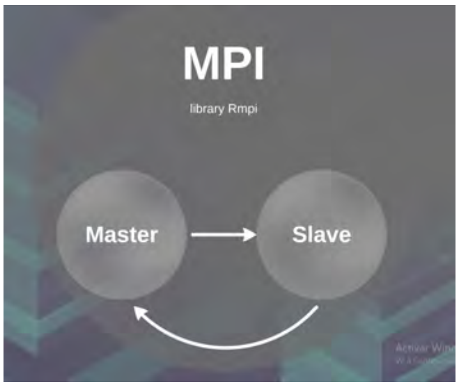


Figure 3: MPI Workflow

The Master file:

In order to implement MPI in our code, we make use of `Rmpi`, a specific library for this purpose. Here we can spawn our processes and check the size (equal to the number of nodes used) using function `Rmpi::mpi.comm.size(0)` the rank of each process, `mpi::mpi.comm.rank(0)` and the name of the processor where each process is located `Rmpi::mpi.get.processor.name()`

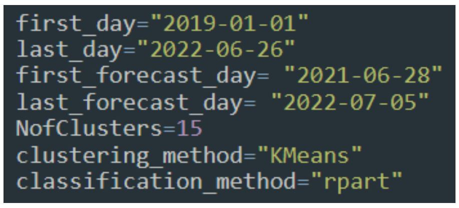
By defining the variable ,N we decide how many ID's each node will compute, so for example, if we use 4 nodes and N = 80, we will be calculating the models for 320 users.

The master file calls the slave file and its respective function, passing as an argument the position of the ID's to compute: $(N*\texttt{rank}+1, N*(\texttt{rank}+1))$.

The Slave file:

In the Slave files is where we actually find our computing code and we have a different slave file for each parallel function (in total 6: `for loop`, `do`, `dopar`, `mcapply`, `parlapply`, `parsapply`).

After defining some of our choice parameters needed for the models:



```
first_day="2019-01-01"
last_day="2022-06-26"
first_forecast_day= "2021-06-28"
last_forecast_day= "2022-07-05"
NofClusters=15
clustering_method="KMeans"
classification_method="rpart"
```

Figure 4: Parameters given to the model

and loading the external files for computing the models and retrieving the data, our only task left is to apply the paralellization code described on Figure 2.

Batch

So now our code is finished we just need to send it to compute. For this we will use a batch file and slurm will be in charge of allocating the computing on the spare nodes.

Our batch file consists of 4 parts. A first part where we say which partition we are using, how much memory we are going to use and the amount of time we want our code to be running for. A second part where we say how many nodes we are going to use and how many tasks per node (with parallelization we will distribute computations across each node, that's why we are only having 1 task per process), a third part where we say where our logout file is going to be sotred and the final part where we load both mpi and R modules and run the script using srun.

We tried computing out code for up to 8 nodes. Because of a problem with the cluster provider, when running for a number of ID's bigger than 300, cluster broke down and we therefore weren't able to collect data (We had to restart cluster on multiple occasions). This problem with the data base is being fixed at the moment so we hope to retrieve more data and run our code for more users.

Results

So as I said, our main goal was to make the code run and by the way comparing the performance of different parallelisation functions. For that, keeping a constant number of data, we can run the code across different number of nodes: 1, 2, 3 or within the same number of nodes we can change the value of N and see how the functions behave.

Firstly, we will with two nodes and vary N: in the following tables and plots we will compare the performance of the creation of Random Forest model.

Table 1: 2 Nodes, N = 20

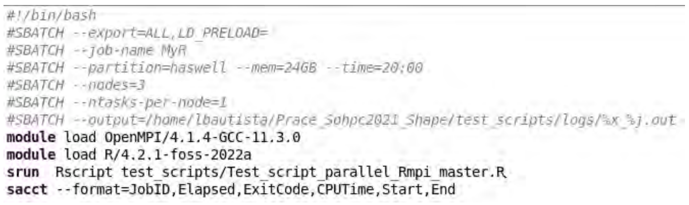
	Average time (seconds)
for bundle	33.032
mcapply	17.078
parSapply	3.993
parLapply	3.971

Table 2: 2 Nodes, N = 40

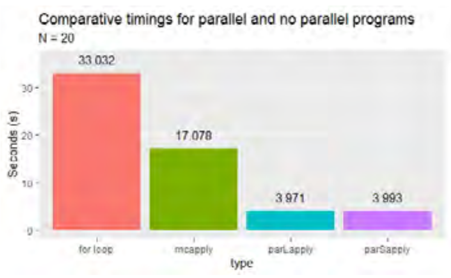
	Average time (seconds)
for bundle	63.704
mcapply	34.887
parSapply	4.803
parLapply	5.206

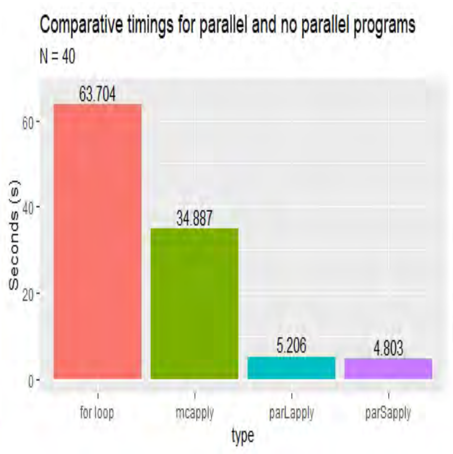
Table 3: 2 Nodes, N = 80

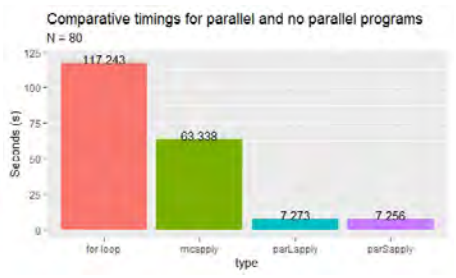
	Average time (seconds)
for bundle	117.243
mcapply	63.338
parSapply	7.256
parLapply	7.273

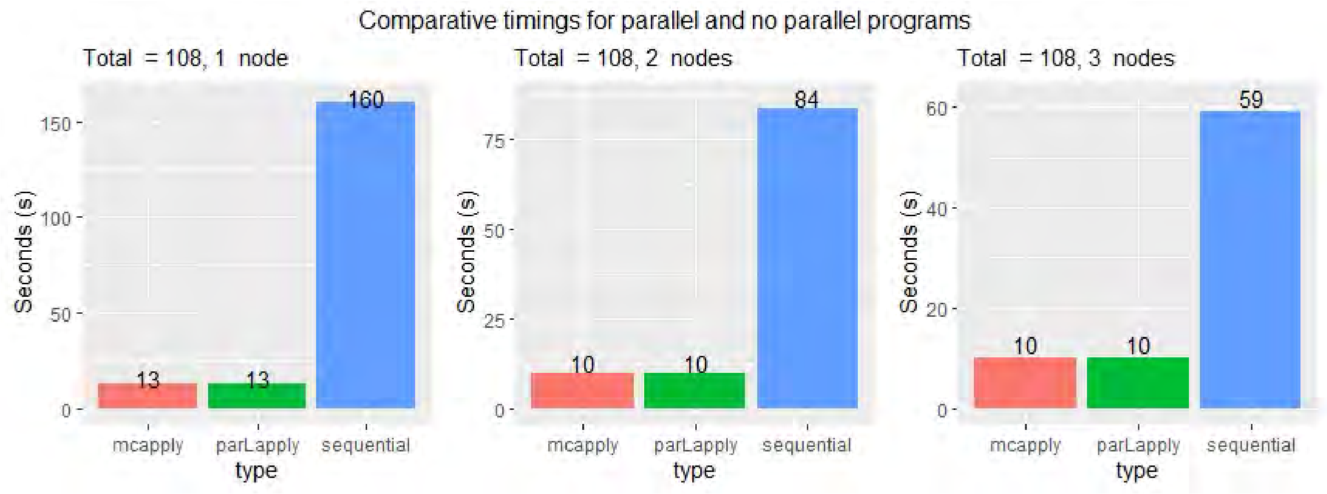


```
#!/bin/bash
#SBATCH --export=ALL,LD_PRELOAD=
#SBATCH --job-name MyR
#SBATCH --partition=haswell --mem=24GB --time=20:00
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=1
#SBATCH --output=/home/lbautista/Prace_Sohpc2021_Shape/test_scripts/logs/%x_%j.out
module load OpenMPI/4.1.4-GCC-11.3.0
module load R/4.2.1-foss-2022a
srun  Rscript test_scripts/Test_script_parallel_Rmpi_master.R
sacct --format=JobID,Elapsed,ExitCode,CPUTime,Start,End
```









Running time for N = 108 on 1, 2 and 3 nodes

In the first graphic we see, as we previously indicated, the comparison for two nodes. The functions that apply parallelisation scale as expected because as we take more ID's the ratio between sequential time and parallel time increases (11, 12, 16 respectively). On the second graphic we are plotting, for N = 108 the timings for different nodes. Here we there are changes when moving from using 1 node to 2 nodes but when we move from 2 nodes to 3 we see that times for parallelisation remain the same. Although it is not represented, when we expand to four nodes the times for these functions remain the same. We suspect that if we increased the number of nodes to 5, 6, 7 the same would happen although we were unable to test it due to a problem with the cluster.

Exact clustering

I will briefly talk about this project because although it was my main goal, it has made me learn a lot about clustering algorithms. The problem was apparently simple. We had an algorithm for the solution to the MSSC clustering problem and our approach was, with different datasets (artificial and real), to apply the algorithm for different numbers of clusters and compare different measures of fit expecting to obtain, for k equal to the ground truth, the best fit.

PRACE SoHPCParallel big data analysis within R for better electricity consumption prediction

Official title of the project

Lucía Absalom Bautista

PRACE SoHPCUniversity of Ljubljana

Ljubljana, Slovenia

PRACE SoHPCAuthors

Lucía Absalom Bautista, [association,] country

PRACE SoHPCMentor

Janez Povh,LECAD, Ljubljana

PRACE SoHPCContact

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

PRACE SoHPCSoftware applied

Virtuoso

PRACE SoHPCMore Information

www.virtouso.org

PRACE SoHPCAcknowledgement

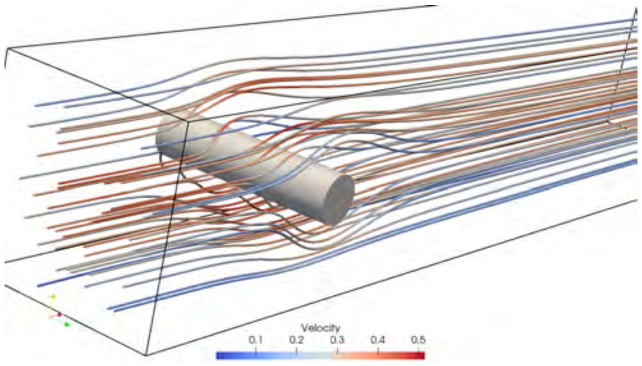
Thanks to my mentor for supporting me and helping me with everything I needed, even when he was on holidays.

PRACE SoHPCProject ID

2218

CFD with FEniCSx

Konstantinos Kellaris



FEniCSx is a powerful computing platform utilizing the Finite Element Method and has shown excellent parallel performance. This project uses FEniCSx to create a solver for the incompressible Navier-Stokes equations and validate it using experimental data.

Computational Fluid Dynamics (CFD) is a branch of fluid mechanics that utilizes numerical methods in order to solve time-dependent partial differential equations (PDE) that dictate fluid flows, specifically the Navier Stokes equations. The numerical approximation of PDEs transforms the continuum equations into matrix equations that must be solved by direct or indirect solvers. Thus, the available computational power dictates both the speed that solutions are obtained as well as the size of the problem one can study.

There are several methods that CFD utilizes for numerical approximation. One of the most common among them is the Finite Element Method (FEM). FEM allows us to approximate the solution of PDEs as a linear combination of (usually polynomial) basis functions that interpolate the solution in small regions of the computational domain (Finite Elements). The original PDE (strong form) that defines the problem at hand and its boundary conditions are multiplied by a test function and integrated throughout the whole problem domain. The resulting equations are called the weak form of the problem.

Then, using calculus of variations the final system of equations is determined and the solution to it are the coefficients of the basis functions that are usually the values of the field in the mesh nodes.

FEniCSx

In this project, the FEniCSx open-source FEM-based computational platform was used, as it shows great efficiency in a parallel architecture setting. FEniCSx is the new version of the legacy FEniCS library. It provides both Python (which was used in this project) and C++ interfaces. It supports distributed memory systems using MPI, allowing the user to write programs in Python which is really user-friendly and handle the parallel execution in a more abstract way. The computational mesh can be generated using the Gmsh Python API, allowing for more control in mesh refinement and the meshing of more complex geometries.

Furthermore, FEniCSx uses the Unified Form Language (UFL) that allows the user to create variational forms required by FEM in a notation system that resembles the actual mathematical notation. The solution of the assembled linear system can be achieved using the PETSc linear algebra suite, which provides access to a wide variety of linear solvers, both direct and iterative, and preconditioners. Finally, the results can be written in parallel to the widely supported XDMF file format and visualized by external programs such as ParaView or VisIt.

Navier-Stokes equations

The developed solver is able to handle both steady-state and transient 3-dimensional incompressible laminar fluid flows. The equations that describe these problems are the incompressible Navier-Stokes equations:

$$\nabla \cdot \mathbf{u} = 0$$

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = \nabla\cdot\sigma\left(\mathbf{u}, p\right) + \mathbf{f}$$

where ρ is the fluid density.

In the above expression, $\mathbf{f}$ denotes the external forces vector, gravity for example. In this application, no external forces are taken into account. Additionally, $\sigma\left(\mathbf{u}, p\right)$ is the stress tensor and is defined as:

$$\sigma\left(\mathbf{u}, p\right) = 2\mu\epsilon\left(\mathbf{u}\right) - p\mathbf{I}$$

where $\epsilon\left(\mathbf{u}\right)$ is the strain-rate tensor:

$$\epsilon\left(\mathbf{u}\right) = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla\left(\mathbf{u}\right)^{\mathrm{T}}\right)$$

and μ is the fluid dynamic viscosity.

The Navier-Stokes equations are non-linear and these terms require spatial treatment. One approach is to solve a non-linear system of equations, and the other one is to utilize prediction correction methods, essentially breaking the system of 2 equations into 3 (or more) that are linear. This solver implements a 3 step prediction-correction scheme called IPCS (Incremental Pressure Correction Scheme). Additionally, spacial care is taken in order for the time steps to satisfy the Courant Friedrichs Lewy condition $(\text{CFL} < 1)$, ensuring solver's stability when using explicit time discretization schemes.

Benchmark

In order to verify the developed solver, a common test case was selected. Specifically, the 3-dimensional laminar flow around a cylinder that lies inside a square channel. The problem definition, results from different solvers and some experimental data are available.[1] The geometry of the problem can be seen in Figure 1, the channel length is $L = 2.5$ m, the height and with of it are $H = 0.41$ m and the cylinder's diameter is $D = 0.1$ m. The fluid's density is $\rho = 1$ kg/m^3 and it's kinematic viscosity is $\nu = 10^{-3}$ m^2/s.

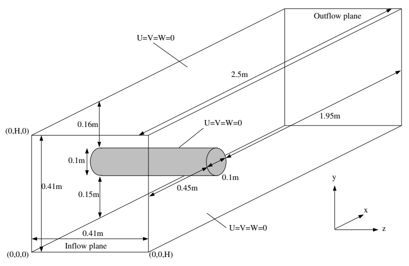


Figure 1: Problem geometry.[1]

The problem boundary conditions are defined as follows:

1. Inlet plane: known parabolic velocity profile and zero pressure gradient
2. Outlet plane: zero gradient velocity and fixed zero pressure value
3. Channel surfaces: zero velocity values and zero pressure gradient, this is known as the no-slip condition
4. Cylinder: same as above

Two different cases were defined, one steady-state and one transient. These cases lead to different inlet velocity profiles. Specifically, the inlet velocity vector is $\mathbf{u}_{in} = (u_{in}, 0, 0)$. For the steady-state case:

$$u_{in} = 16 U_m yz \frac{(H-y)(H-z)}{H^4}$$

where $U_m = 0.45$ m/s. This velocity profile corresponds to a mean velocity magnitude of $\overline{U} = 0.2$ m/s. The Reynolds number of the flow is defined as:

$$\text{Re} = \frac{\overline{U} D}{\nu}$$

The Reynolds number is a non - dimensional number that represents the ratio of inertia forces to viscous forces acting on the fluid flow. Low Reynolds numbers correspond to laminar flow. For the steady case, $Re = 20$.

The inlet velocity profile for the transient case is also parabolic, but it is time-dependent:

$$u_{in} = 16 U_m yz \frac{(H-y)(H-z)}{H^4} \sin(\pi t/8)$$

where $U_m = 2.25$ m/s. The total simulation time is $T = 8$ s. In this case the Reynolds number is also time-dependent with values in the range $0 \le \text{Re}(t) \le 100$, inside the laminar flow regime.

For these cases, several data are available.[1,2] Firstly, the drag and lift forces applied to the cylinder are defined below:

$$F_D = \int_S \left(\mu \frac{\partial \mathbf{u_t}}{\partial \mathbf{n}} n_y - p n_x \right) dS$$

$$F_L = - \int_S \left(\mu \frac{\partial \mathbf{u_t}}{\partial \mathbf{n}} n_x + p n_y \right) dS$$

where S is the cylinder surface, $\mathbf{n}$ is the normal surface vector, $\mathbf{t} = (n_y, -n_x, 0)$ is the tangent vector and $\mathbf{u_t} = \mathbf{u} \cdot \mathbf{t}$ is the tangential velocity vector on S. Consequently, the drag and lift coefficients are defined:

$$C_D = \frac{2 F_D}{\rho D H \overline{U}^2}$$

$$C_L = \frac{2 F_L}{\rho D H \overline{U}^2}$$

Additionally, a useful metric is the pressure difference between a point in the front and a point in the back of the cylinder. These points lie at the intersection of 2 symmetry planes of the channel (along the Y and Z axes) with the cylinder.

Mesh

After defining the test cases, the next step was to create a mesh. The computational domain was discretized using the Gmsh Python API, creating an unstructured computational mesh, that contains only tetrahedral cells. As shown in Figure 2, the mesh has been refined in a region around the cylinder and the walls near it.

Figure 2: Mesh generated with Gmsh.

In order for the simulations to run in parallel, the mesh should be decomposed. The way a mesh is decomposed and distributed to processors affect the quantity of simulation time spent in communication between processors. Here, the MPI-based ParMETIS library was used.

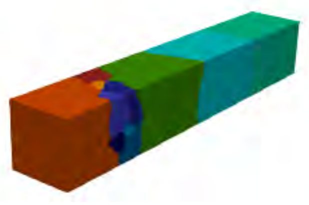


Figure 3: Mesh decomposed in 16 processors.

Results

Firstly, the steady-state case was solved for 6 different setups. The simulations were run on the Iris Cluster of the university of Luxembourg using 64 up to 360 processors. Three different meshes were investigated: 68.342, 528.253, and 1.759.240. Furthermore, 2 different polynomial order combinations were used: 2nd order for velocity and 1st for pressure (V2-P1) or 3rd order velocity and 2nd pressure (V3-P2). Then, the results were compared to the reference values. As is evident from Figures 4, 5, 6, the solver manages to correctly solve the flow and its accuracy is improved as mesh resolution is increased.

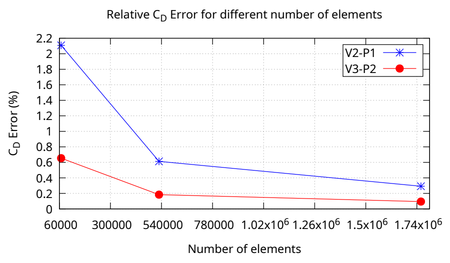


Figure 4: C_D relative errors for steady-state case.

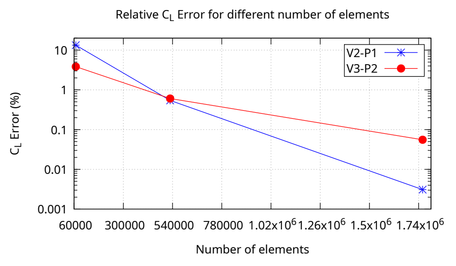


Figure 5: C_L relative errors for steady-state case.

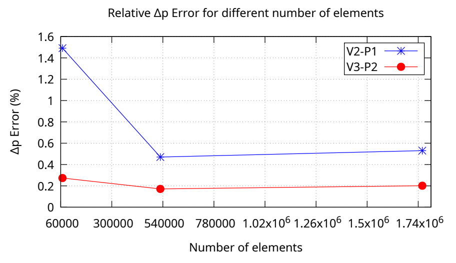


Figure 6: ΔP relative errors for steady-state case.

Furthermore, the computational cost in core hours is presented in Figure 7, one can observe the difference in magnitudes of cost for finer meshes as well as simulations using different polynomial orders.

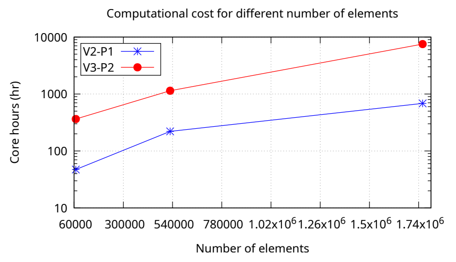


Figure 7: Computational cost for different mesh sizes.

In Figures 8 and 9 the velocity contours are shown in 2 channel symmetry planes.

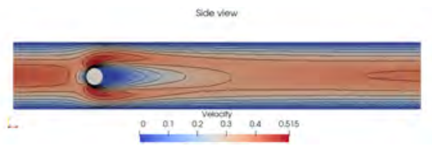


Figure 8: Velocity magnitude Z axis mid-plane for steady-state case.

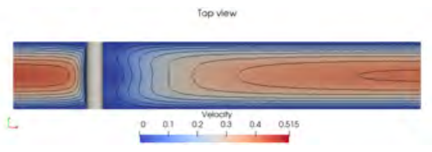


Figure 9: Velocity magnitude Y axis mid-plane for steady-state case.

Then, the transient case was simulated using a constant time-step $\Delta t = 2 \cdot 10^{-4}$ s. Three different cases were simulated and they are shown in Table 1.

Table 1: Transient cases setup.

	# of elements	pol. orders
Case 1	145.608	V2-P1
Case 2	145.608	V3-P2
Case 3	1.091.160	V2-P1

The simulations were run on the Iris Cluster using 144 processors for Case 1 & 2 and 360 processors for Case 3.

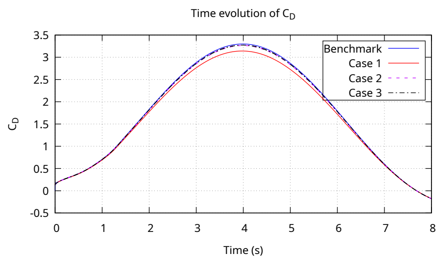


Figure 10: C_D time series of transient cases.

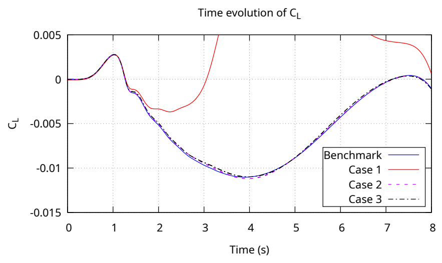


Figure 11: C_L time series for transient cases.

Figures 10 and 11 show the time evolution of the drag and lift coefficients. All 3 cases match the reference C_D time evolution, but case 1 fails to predict the lift. This is an interesting behavior, showing us that the most important metric in order to improve a solver's accuracy is the total degrees of freedom, the product between the number of elements and degrees of freedom per element.

Discussion & Conclusion

In this work, a solver for 3-dimensional laminar incompressible fluid flows using the FEM method was successfully developed on the FEniCSx platform and then validated with experimental data. Furthermore, the effect of mesh size and order of polynomial basis functions to the solver's accuracy was investigated. The results are encouraging, showing acceptable accuracy even for intermediate mesh density without disproportional computational costs.

These studies validated the solver but revealed some aspects that it can be extended and improved.

Firstly, the code can be extended in order to use meshes generated from more user-friendly external meshing software, such as ANSYS Meshing.

Furthermore, the current time marching scheme can be adjusted in order to use adjustable time-step methods, in order to save on computational costs. This approach requires a careful selection of stability conditions, but the code already provides interfaces for that.

Finally, the code can be extended to handle turbulent flows using the Reynolds Averaged Navier Stokes (RANS) equations paired with a turbulent closure model, e.g $k - \epsilon$.

References

[1] Schäfer, M., Turek, S., Durst, F., Krause, E., Rannacher, R. (1996). Benchmark Computations of Laminar Flow Around a Cylinder. In: Hirschel, E.H. (eds) Flow Simulation with High-Performance Computers II. *Notes on Numerical Fluid Mechanics (NNFM)*, 48. Vieweg+Teubner Verlag. DOI: 10.1007/978-3-322-89849-4_39.

[2] Bayraktar, E., Mierka, O., Turek, S. (2012). Benchmark computations of 3D laminar flow around a cylinder with CFX, OpenFOAM and FeatFlow. *International Journal of Computational Science and Engineering*, 7(3), 253-266. DOI: 10.1504/IJCSE.2012.048245.

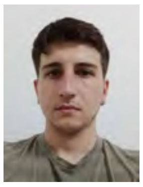
PRACE SoHPCProject Title
Computational Fluid Dynamics

PRACE SoHPCSite
ULux - Université du Luxembourg, Luxemburg

PRACE SoHPCAuthors
Konstantinos Kellaris, National Technical University of Athens, Greece

PRACE SoHPCMentor
Dr. Ezhilmathi Krishnasamy, ULux, Luxembourg

PRACE SoHPCContact
Konstantinos, Kellaris, N.T.U.A.
E-mail:
kellariskonstantinos@gmail.com



Konstantinos Kellaris

PRACE SoHPCSoftware applied
FEniCSx, Gmsh, ParaView

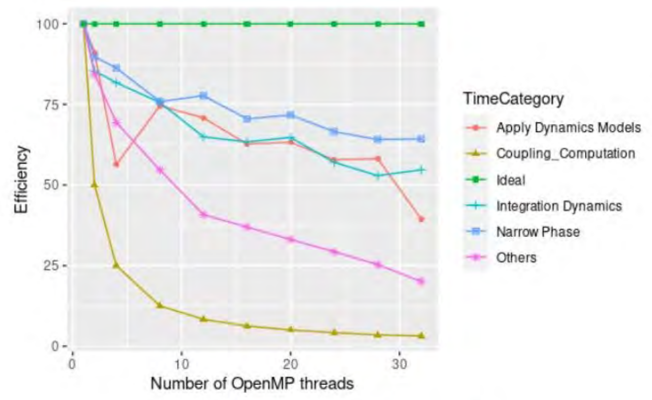
PRACE SoHPCMore Information
fenicsproject.org
gmsh.info
www.paraview.org

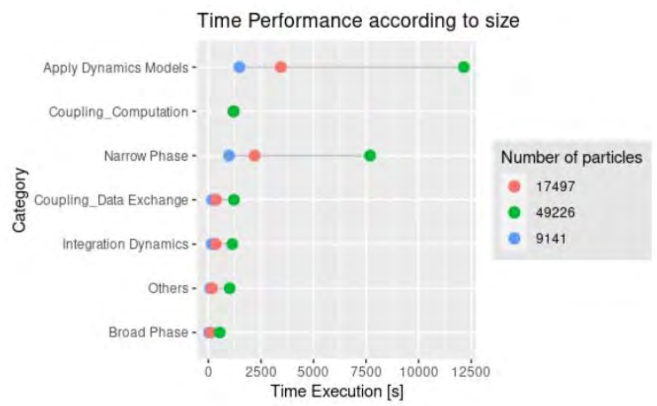
PRACE SoHPCAcknowledgement
I would like to thank my mentor Dr. Ezhilmathi Krishnasamy for providing guidance during the project. I would also like to thank PRACE for providing me with this great opportunity.

PRACE SoHPCProject ID
2219

Scalability, performance comparison and regression of XDEM Multi-Physics simulations

Performance report for XDEM software





Alexander Trujillo

Performance regression and continuous integration are crucial steps to develop a scientific software, in this case XDEM. This project aims to compare the performance between multiple configurations or versions, additionally with a automation process to obtain these results in a cluster environment.

Maybe you heard this phrase ... “There is no free lunch!”, and this could no be more truer when it is applied to scientific software development, specially taking in mind the exploitation of computational resources available to the maximum to obtain the precious results. But sadly it does not end there, it sounds like common sense that your simulations will run faster if you keep adding more processors to your current resources, but believe it or not, that could make it worse at the end of the day.

To explain this situation, if we want to exploit many processors, we need to be creative and come up with a parallel implementation and divide the workload. So we can imagine this as ten people building ten houses, generally is better idea that each people build one house concurrently than those 10 people working on one single house and then to the rest, but it is likely that one person has no task to do, or worse if one could hinder another one work.

The aim of the project is to develop R and bash scripts to point to the changes in performance or scalability for different configurations, problem sizes or versions. Additionally, we move forward to automatize this project using Aion cluster by SLURM (a software used to manage resources and schedule jobs), in this way we gather data from different versions or configurations and obtain the plots pointing the critical changes on the performance.

XDEM Software

The XDEM software[1] is a numerical simulation framework implemented using C++, based on extended discrete element method, a method which besides describing the dynamics of granular material by means of spatial and temporal information, also obtains additional properties of the conglomerate of particles as thermodynamic state , stress, etc. It could even be coupled with CFD solver as OpenFOAM to account for liquid or gaseous phases around the particles.

This implementation follows an hybrid parallelization approach, this means is capable to run using multithreading on a single compute node (OpenMP) and across multiple nodes using MPI.[2]

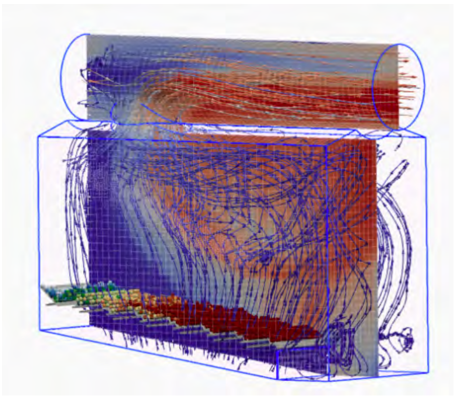


Figure 1: Biomass furnace simulation

XDEM can simulate complex physical processes such as biomass furnace,[3] shown in Fig 1.

Motivation of this project

The scripts will be part of XDEM Continuous Integration which is important to ensure and maintain the performance of the software with new versions of the code, we automate the process of performance analysis and regression detection. In this way developers could check if the changes affect the performance of the simulation, and highlight the part of the computation which is impacted.

How XDEM does this simulations?

XDEM has many computational phases for each simulation case but 4 are the most important ones and usually those take the most of the execution time in total[4].[5]

- **Broad phase**: A preliminary approximation that builds a list of particle pairs that can potentially interact. To make a fast calculation, particles are represented by bounding spheres.
- **Narrow phase**: Now performs a precise contact detection using the actual shape of the particle and calculates the actual distance between particles.
- **Apply dynamics model phase** : Related to solve the dynamics models that is defined for each particle (impact, bonding, rolling ,etc) and as well calculates the contribution of the interaction to each particle involved.
- **Integration phase**: Takes care of updating the state of the particles according to the neighbors contributions.

Decomposition and communication

A important concept is domain decomposition, which is how workload is distributed throughout the compute nodes, the spatial location is divided in cell grids, then cells will be grouped in subdomain and assigned to a certain compute node, this last process vary depending of the decomposition algorithm but in a broad sense it converts these cells into nodes that belongs to a graph and performs the partitioning according to the neighbor relationships with the objective of balancing the workload and minimizing the communication.

Once the decomposition is done, we create the boundary of another compute nodes as these need to share information of the particles constantly in run time, these cells are called ghosts layers. As XDEM uses a MPI parallelization paradigm, there is a latency between these layers and communication is bounded by the bandwidth of the network.

R, a language made for this

The data loading and transformation will be possible thanks to R, which is a programming language mostly present on statistics analysis and graphics.
R is very well maintained thanks to the open-source community and contains thousand of libraries implemented by developers around the world that make your life easier.
We based our work on “tidyverse” as the most important package library, it allow us to extract transform the data present on our simulations, as well as a intuitive set of commands to manipulate the data structure to be more suitable for different kind of plots. Also "ggplot2" which offers a great variety of plots and graphs implementations according to the need to convey the right information to the viewer, as one of the most reliable sources to select the kind of plot suitable for our project was `https://www.data-to-viz.com/`
With these scripts we can compare directly the results for different parameters and can quickly notice if there is a efficiency problem.

Automation with bash and SLURM

Bash, a command language which main task so to speak is to process the commands you give to it, from simple things as show “Hello world” on your terminal to complex scripts which manages several files. One example of usage is when you want to launch an specific executable, if you developed your own software you for sure want to test as soon as you finish it or in the meantime to see if it is working as intended, you can execute it in a bash script and make it suitable for your needs, do you want it to run thousand of time for different arguments or cases? No prob!, bash as well can manage for loops so you can automate this process.

Bash is a UNIX shell, besides the best known ones are Mac OS and Linux, and guess what, more than 95% of supercomputers in the world uses Linux.

Normally a supercomputer is a shared resource between several scientists and because of that, exists certain policies, priorites or amount of restricted resources available for each user. This supercomputer must contain a software that acts like a orchestra conductor, organizing the submitted jobs taking into account the priorities and policies, then scheduling the resources for everyone, in this case AION uses a software called SLURM which will not let this supercomputer fall into anarchy.

AION specifications

The current project will make use of a supercomputer located at University of Luxembourg, called AION which consists of 318 compute nodes with 256Gb of RAM and 128 cores each one, totaling 40704 compute cores, with a peak performance of about 1,70 PetaFLOP/s. This means it can 1.7 quadrillion arithmetic operations per second at its best.

Results

Dataset used

The data acquired to test the performance report is from a blast furnace charging simulation, this involves intricate physical phenomena that must be modeled (and validated) carefully and require the use of High-Performance Computing (HPC) platforms. Here in this following `https://www.youtube.com/watch?v=7Va4ALbm9Y8` we can observe the graphical environment for this simulation.

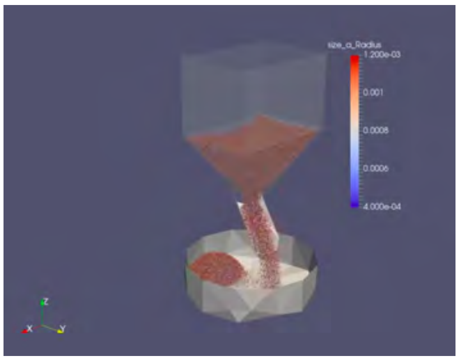


Figure 2: Test case: Blast furnace charging simulation

Strong Scalability

Shows the time execution evolution according to the number of compute nodes, as well as the speedup plot. The R script loads and transform the data generated by XDEM and once obtains the time execution for the different phases, it creates a speedup plot for each category and focusing on the most relevant (the costliest categories). A heatmap of the efficiency accross different categories is included as well as a steamgraph of the proportion of execution time to observe the evolution and change in importance of the differente categories when more compute nodes are added.

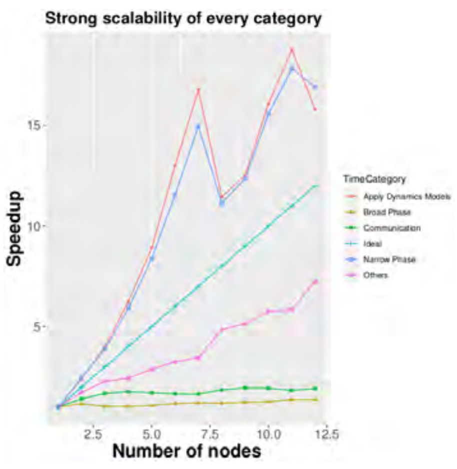


Figure 3: Speed-up plot for each category

Weak Scalability

The featured plots for the weak scalibity report are similar for the strong scalalibility case, but in these case it is required by the script to explicitly state the number of particles or size used for each run.

Complexity characterization

The analysis done here identifies the complexity of the measured computing time for the principal categories as well as evolution graph of the time execution for the categories. Besides it creates a heatmap for the scaling factor, checking as reference a linear increase, if it is scaling in a higher order it reveals a deficiency on the category.

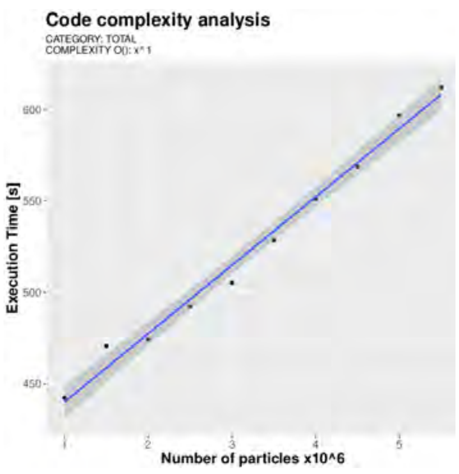


Figure 4: Linear trend found for the blast furnace charging simulation

Performance regression

Mainly compares the performance change between a reference and different software versions.
It shows the drop or rise in percentage for total execution or for each computational phase. As well as a heatmap pointing out the worst and better performant category for each version.

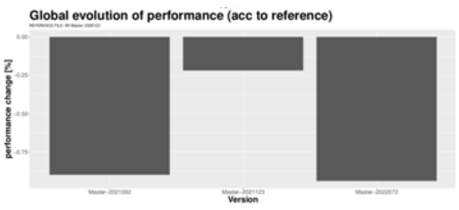


Figure 5: Barplot pointing the performance changes between multiple versions

Decomposition comparison

It creates individual evolution plots for each decomposition compared to a reference (in this case a ORB decomposition), pointing out the change in execution time for each computational phase

Workload balance report

For the workload balance creates a total imbalance plot for the different compute ranks, showing if there is an uneven distribution. Also it shows the load imbalance factor which is the ratio of the maximum imbalance and the average, meaning a factor close to 1 is a even distributed workload.

Conclusions

Now XDEM has an additional tool that will help this software to keep track of the improvement over time, developers could integrate it to continuous integration in the GitHub workflows.
Even users could test with a little version of the original work to see which is the best decomposition or version for their project.
For future work could be interesting to add an extra tool to obtain the average of several runs to avoid extreme variability on the execution time for certain simulations. As well as a database linkage to keep track of previous performance analysis.

References

[1] Bernhard P. (2020). Advanced Multi-physics Simulation with XDEM Software https://luxdem.uni.lu/software/index.html

[2] Checkaraou, A., Rousset, A., Besseron, X., Varrette, S., & Peters, B. (2018). Hybrid MPI+openMP Implementation of eXtended Discrete Element Method. 2018 30Th International Symposium On Computer Architecture And High Performance Computing (SBAC-PAD). doi: 10.1109/cahpc.2018.8645880

[3] Besseron, X., Rusche, H., & Peters, B. (2022). Parallel Multi-Physics Simulation of Biomass Furnace and Cloud-based Workflow for SMEs. Practice And Experience In Advanced Research Computing. doi: 10.1145/3491418.3530294

[4] Chekaraou, A., Besseron, X., Rousset, A., Kieffer, E., & Peters, B. (2020). Predicting near-optimal skin distance in Verlet buffer approach for Discrete Element Method. 2020 IEEE International Parallel And Distributed Processing Symposium Workshops (IPDPSW). doi: 10.1109/ipdpsw50202.2020.00093

[5] Besseron, X. (2022). High-Performance Computing for the simulation of particles with the Discrete Element Method. Marseille: CEMRACS 2022. Retrieved from https://hdl.handle.net/10993/51734

PRACE SoHPCProject Title
Performance Comparison and Regression for XDEM Multi-Physics Application

PRACE SoHPCSite
University of Luxembourg, Luxembourg

PRACE SoHPCAuthors
Alexander Trujillo, [ICTP,] Italy

PRACE SoHPCMentor
Besseron, Xavier, Univ. Luxembourg, Luxembourg

W Alexander Trujillo

PRACE SoHPCSoftware applied
XDEM

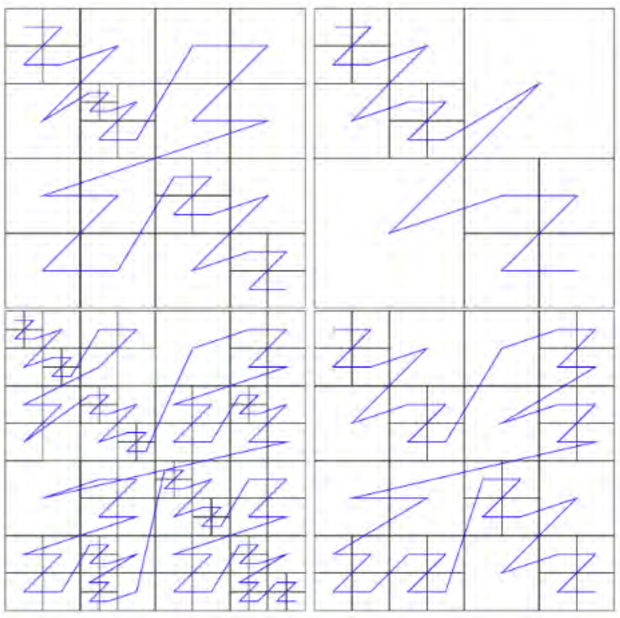
PRACE SoHPCMore Information
https://luxdem.uni.lu/software/index.html

PRACE SoHPCAcknowledgement
I am truly thankful to PRACE program along with Xavier Besseron, Ezhilmathi krishnasamy and Leon Kos which made possible this opportunity.

PRACE SoHPCProject ID
2220

Designing Scientific Applications on GPUs

Filippo Barbari



This project aims to develop a CUDA implementation of the Recursive Sparse Blocks library for sparse matrix multiplications in order to further optimize these common operations.

Abstract

The LibRSB library implements a parallel and cache-efficient way to deal with sparse matrices operations through a unique and hybrid matrix storage format called Recursive Sparse Blocks. Since it's already implemented with the shared memory parallelization of OpenMP and it has proven its performance in many papers, it would be really beneficial to implement a CUDA parallel version of the algorithm.

Introduction

Around 2010, dr. Michele Martone (from the University of Tor Vergata, Rome, Italy) invented the Recursive Sparse Blocks format and implemented it in the LibRSB library. With this work, he tried to tackle the problem of handling really big sparse matrices inside parallel algorithms.

The most common sparse matrix operations, that LibRSB implements, are:

- **SpVV**: a multiplication between a sparse vector and a dense vector
- **SpMV**: a multiplication between a sparse matrix and a dense vector
- **SpMM**: a multiplication between a sparse matrix and a dense one
- **SpSV**: the resolution of a linear system represented by a sparse matrix and a dense vector of known terms
- **SpSM**: the resolution of a linear system represented by a sparse matrix and a dense matrix of known terms

My work has mostly focused on the translation of the SpMV algorithm into a CUDA kernel.

The RSB format

Recursive Sparse Blocks[1] is a hybrid format because it uses three different formats/data structures at three different levels to optimize the hit/miss ratio of the cache:

- at the *root level*, the highest level, submatrices/blocks are stored by using a **Z-Morton sorting**
- at the *intermediate level*, each submatrix/block is recursively subdivided in a **Quad-tree** fashion
- at the *leaf level*, the lowest level, each submatrix/block is stored using one of the common **COO/CSR/CSC sparse matrix formats**

The **Z-Morton sorting** is a sorting method used for keeping the spatial locality of bidimensional data. Z-Morton is a space-filling curve, like the more famous *Hilbert's spiral*, which takes its name from the "Z" shape it has. In the figure 1, you can see some examples of this special sorting method used on the blocks of a recursively subdivided matrix. In particular, the four subdivisions you can see in figure 1 are two equal matrices (the top two and the bottom two) that have been subdivided according to the cache size of two different machines (the left two belong to a 1MB-cache machine, the right two belong to a 2MB-cache machine).

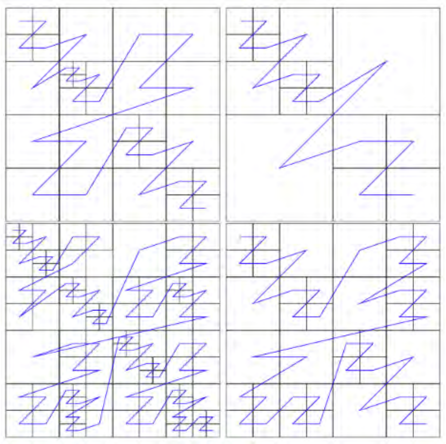


Figure 1: Examples of Z-Morton sorting of blocks in a recursively subdivided matrix.

The **Quad-tree** is a quaternary tree often used to recursively subdivide a continuous bidimensional space in order to optimize the *bidimensional range query*. In the figure 2, we can see an example of a Quad-tree constructed on a continuous bidimensional space. The RSB format uses it to look for the closest non-zero elements while performing a multiplication.

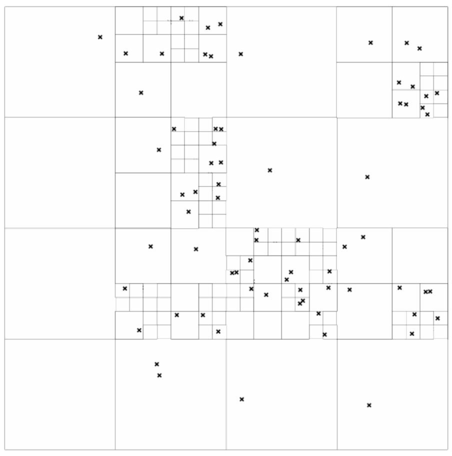


Figure 2: Example of a Quad-tree in a continuous bidimensional space.

This data structure also helps optimize the memory usage since it gives more accuracy only inside those blocks which have more non-zero elements, meaning that they contain more information.

The **COO** format was the first sparse matrix storage format, and also the simplest, to be invented. It's a bad acronym for **COOrdinate list** because it simply stores a sparse matrix as a list of triplets, each one containing exactly three elements:

- the index of the row where the element is
- the index of the column where the element is
- the value of the element

For example, the matrix shown in figure 3 would be stored as shown in figure 4.

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 1 \\ 0 & 0 & 5 & 0 & 0 \end{pmatrix}$$

Figure 3: Example of a sparse matrix.

row idx	0	1	2	3	3	4
col idx	2	3	1	1	4	2
value	3	7	2	6	1	5

Figure 4: COO representation of the matrix in figure 3.

The **CSR** and **CSC** formats are improved versions of the COO format. Their acronyms stand for Compressed Sparse Rows/Columns and they further reduce the space required to store a sparse matrix by *compressing* the contiguous adjacent row indices (column indices for CSC) and, thus relying on the column indices to precisely identify each non-zero element. For example, the matrix in figure 3 would be represented in CSR format as shown in figure 5 and in CSC format as shown in figure 6.

row idx	0	1	2	3	5	
col idx	2	3	1	1	4	2
value	3	7	2	6	1	5

Figure 5: CSR representation of the matrix in figure 3.

col idx	0	0	2	4	5	
row idx	2	3	0	4	1	3
value	2	6	3	5	7	1

Figure 6: CSC representation of the matrix in figure 3.

Note that we needed one less row/column index by using these improved storage formats.

Methods

As I mentioned earlier, my work has mostly focused on the translation of the SpMV algorithm into a CUDA kernel and on the conversion of the matrix allocation functions. I only managed to fully convert only one of the many allocation functions that LibRSB offers, which is the function that converts a COO-formatted matrxi into an RSB-formatted one, into CUDA code by using the Unified Memory.

Results

Unfortunately, at the time of this writing, there's a memory issue in my CUDA kernel implementation and, therefore, I am not able to show you any results.

Conclusion

After all the work I've done on the LibRSB library, I can safely say that it has really great potential because of the wide range of possible use case scenarios (like recommendation systems and network/graph theory) and it has proved its performance and efficiency. The code is made of 400'000 lines of C, C++ and Fortran code divided in 1000 files so extending it would require much more time and manpower.

References

[1] Martone, M. and Filippone, S. and Tucci, S. and Paprzycki, M. and Ganzha, M. (2010). Utilizing Recursive Storage in Sparse Matrix-Vector Multiplication-Preliminary Considerations.

PRACE SoHPCProject Title
Designing Scientific Applications on GPUs

PRACE SoHPCSite
University of Luxembourg, Luxembourg

Filippo Barbari

PRACE SoHPCAuthors
Filippo Barbari, University of Bologna, Italy

PRACE SoHPCMentor
Ezhilmathi Krishnasamy, University of Luxembourg, Luxembourg

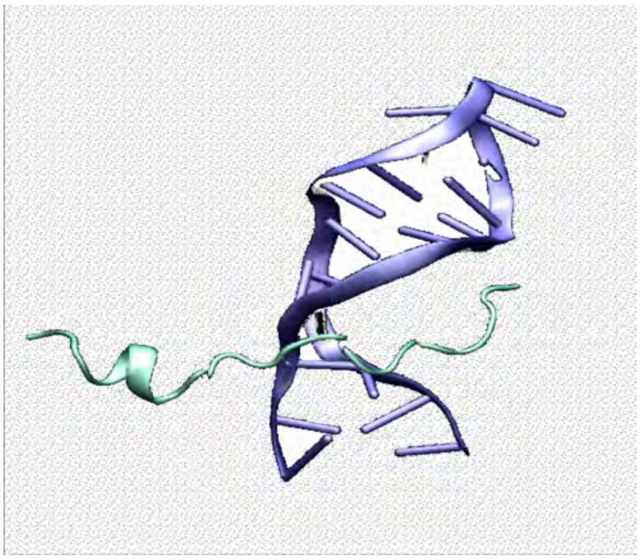
PRACE SoHPCContact
Filippo, Barbari, University of Bologna
Phone: +39 327 220 7037
E-mail: filippo.barbari@gmail.com

PRACE SoHPCAcknowledgement
I would like to thank dr. Ezhilmathi Krishnasamy, my mentor, from the University of Luxembourg, for his great patience and for all the help he has provided me throughout the project.

PRACE SoHPCProject ID
2221

Designing Antiviral Drugs with HPC

Ezgi SAMBUR



Here, by making use of HPC, a drug design study is carried out to block viral ds-DNA, which is associated with many viruses, including HIV/AIDS and HPV. GPUs are used to accelerate MD simulations and MM-GBSA/MM-PBSA calculations are performed within the scope of HPC.

A particular class of well tolerated drugs is formed by small peptides because their building blocks are simply the 20 naturally occurring amino acids — the most fundamental constituents of any living cell. In order to transform such peptidic compounds into effective medical drugs high affinity binding to the target must be accomplished (and is usually the outcome of a tedious and long-lasting optimization process).

Introduction

In this project we aim to develop a peptidic drug against viral ds-DNA, which is associated with Herpes Simplex Virus, Cytomegalovirus, Adenovirus type 5, SV40 Polyoma virus and Vaccinia virus. If viral DNA is blocked it cannot replicate inside the cell nor express viral proteins promoting the degree of infection.

We start from the well-known TAT protein in HIV-1 for which an NMR structure is available detailing the complex geometry between nucleic acid and TAT binding domain.[1] I24 is a 9-mer amino acid sequence that can be fused to TAT thereby enhancing antiviral activity.[2] Here we build on this previous study and insert single-point random mutations in the sequence of TAT-I24. New mutant peptide sequences are subjected to MD simulations (250 ns). An example for a structural snapshot of such a single-point mutation is shown in Fig 1.

After the MD simulations MM-GBSA and MM-PBSA binding free energies[3] are calculated. Peptides with better-than-previous MM-PBSA and MM-GBSA scores can be proposed as new drug candidates and further confirmed in wet-lab experiments.

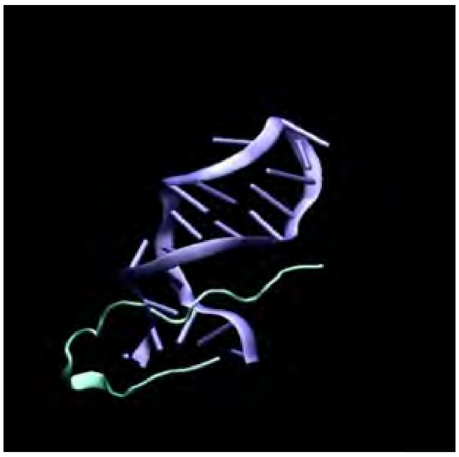


Figure 1: Structural snapshot of mutant TAT-M178 taken from MD simulation.

Methods

AMBER/20[4] was used throughout. Given the rather extended target length of 250 ns of MD simulation, it was essential to employ MD programs of maximum compute performance. As seen in Fig 2 the GPU implementation of *pmemd* — the major MD code in AMBER — gave sufficient levels of compute performance.

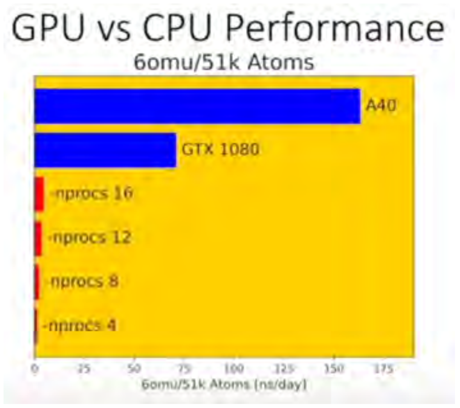


Figure 2: Computational Performance: AMBER/20 benchmark for MD simulations of protein 6OMU.pdb in explicit water ($\approx$51k atoms). Red bars are MPI parallel runs while blue bars represent the achievable performance using a single GPU of type NVIDIA GTX 1080 or A40.

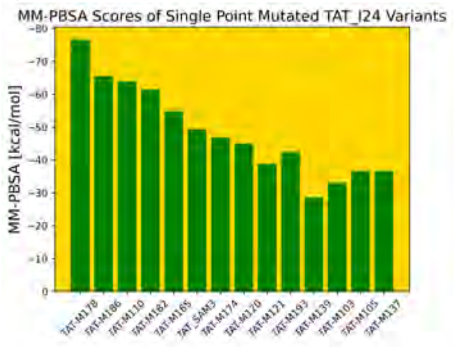
A set of 120 candidate peptides was formed from inserting random mutations into the TAT-I24 sequence of the wt. In a small subset of these constructs deletions were considered instead of single point mutations (either TAT or I24 box). Each of these mutant peptides became subject to long term MD simulation in explicit water (250 ns) starting from the wt structure with modified amino acid sequence. This way individual peptides were given sufficient time to evolve into their native state of complex geometry between peptide and ds-DNA provided as helical hairpins made of GC base pairs.[5]

Upon termination of the MD simulations the final 50 ns of individual trajectories were taken into account for binding free energy calculations, ΔG, following the MM-PB(GB)SA approach.[3] Here the reaction $R + L \overset{\Delta G}{\rightleftharpoons} R/L$ is quantified with R the receptor (ds-DNA in our case), L the ligand (TAT-I24 variants in our case) and R/L the complex formed from free non interacting reactants, ie isolated R and L in aqueous solution. The equilibrium will be shifted to the product site — hence the complex will be formed strongly — the larger value of negative sign we obtain for ΔG. Thus we can conveniently rank all of the 120 TAT-I24 variants because the reference state of the wt was previously estimated to amount to ΔG_{wt}=-36 kcal/mol.[2]

Results

Top ranked candidates for new peptidic drugs enhancing the affinity to ds-DNA are summarized in Fig 3. Both, MM-PBSA (Fig 3a) as well as MM-GBSA results (Fig 3b) are included. The list comprises more than 10 peptides with significantly improved binding affinity when compared to the wt, some of them even showing an impressive shift in ΔG. Similar to previous observations[2] MM-GBSA results tend to display larger ΔG values (of negative sign) when compared to MM-PBSA results. TAT-M178 was the highest ranked new drug candidate to substitute TAT-I24(wt) with an associated ΔG_{PB} of -77 kcal/mol.

a)



b)

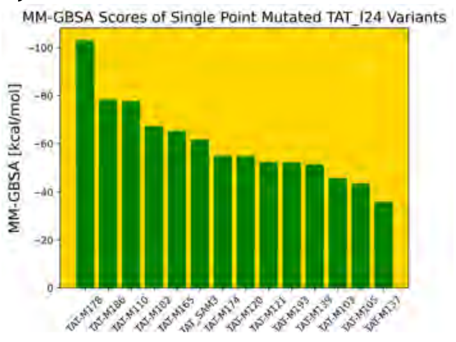
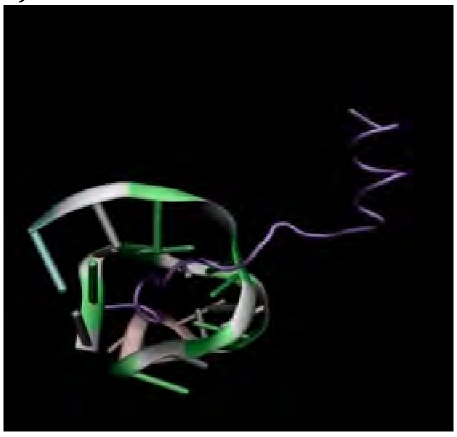


Figure 3: Top ranked mutant peptides with corresponding binding free energies obtained from MM-PBSA analysis (a) or MM-GBSA analysis (b).

In addition to bare thermodynamic assessment, the underlying MD simulations can also give structural insights into the binding process. For example at the start of the MD simulation of TAT-M178 there is clearly a lack of tight interactions between ds-DNA and the peptide as visible from the detached α-helical stretch to the right (Fig 4a). Only at later stages overall strong interaction between the entire peptide and ds-DNA can be established (Fig 4b).

a)



b)

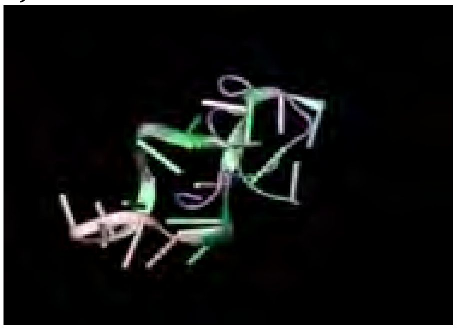


Figure 4: Early (a, frame 0) and late (b, frame 100) structural snapshot in the MD trajectory of TAT-M178.

References

[1] Pham, V.V. et al. (2018) doi:10.1038/s41467-018-06591-6

[2] Harant, H. et al. (2021) doi:10.3390/biologics 1010003

[3] Sim, S. et al. (2017) doi:10.1002/1873-3468.12574

[4] Case, D.A. et al. (2020) AMBER 2020, University of California, San Francisco

[5] Hsu, C.H. et al. (2005) doi:10.1093/nar/gki725

PRACE SoHPCProject Title

HPC-Derived Affinity Enhancement of Antiviral Drugs

PRACE SoHPCSite

VSC Research Center, TU Wien, Austria

PRACE SoHPCAuthor

Ezgi SAMBUR, Bahçeşehir University, Istanbul, Turkey

PRACE SoHPCMentor

Siegfried Hoefinger, VSC Research Center, TU Wien, Austria

PRACE SoHPCCo-Mentor

Markus Hickel, VSC Research Center, TU Wien, Austria

PRACE SoHPCContact

Ezgi, Sambur, Health Sciences Institute, Bahçeşehir University Istanbul, Turkey
Phone: +09 5462354991
E-mail: ezgisambur@gmail.com

PRACE SoHPCProject ID

2222

Ezgi SAMBUR

www.summerofhpc.prace-ri.eu