PRACE

PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE


summerofhpc.prace-ri.eu

A long hot summer is time for a break, right? Not necessarily! PRACE Summer of HPC 2016 reports by participants are here.

# HPC in the summer?

*Leon Kos*

There is no such thing as lazy summer. At least not for the 20 participants and their mentors at 10 PRACE HPC sites.

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

This year, training week was at Juelich and it seems to have been the best training week yet! From MPI to Go-karting and good food, the week was a blast! It was a great start to Summer of HPC and set us up to have an amazing summer!

At the end of the summer videos were created and are available on Youtube as PRACE Summer of HPC 2016 presentations playlist. Together with the following articles interesting code and results are available. Dozens of blog posts were created as well. At the end of the activity, every year two projects out of the 20 participants are selected and awarded for their outstanding performance. The winners of this year, Anurag Dogra and Marta Cudova, presented their experience at the award ceremony in Cineca supercomputing centre, Italy.

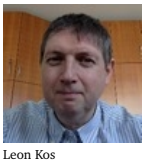
Therefore, I invite you to look at the articles and visit the web pages for details and experience the fun we had this year.

## Contents



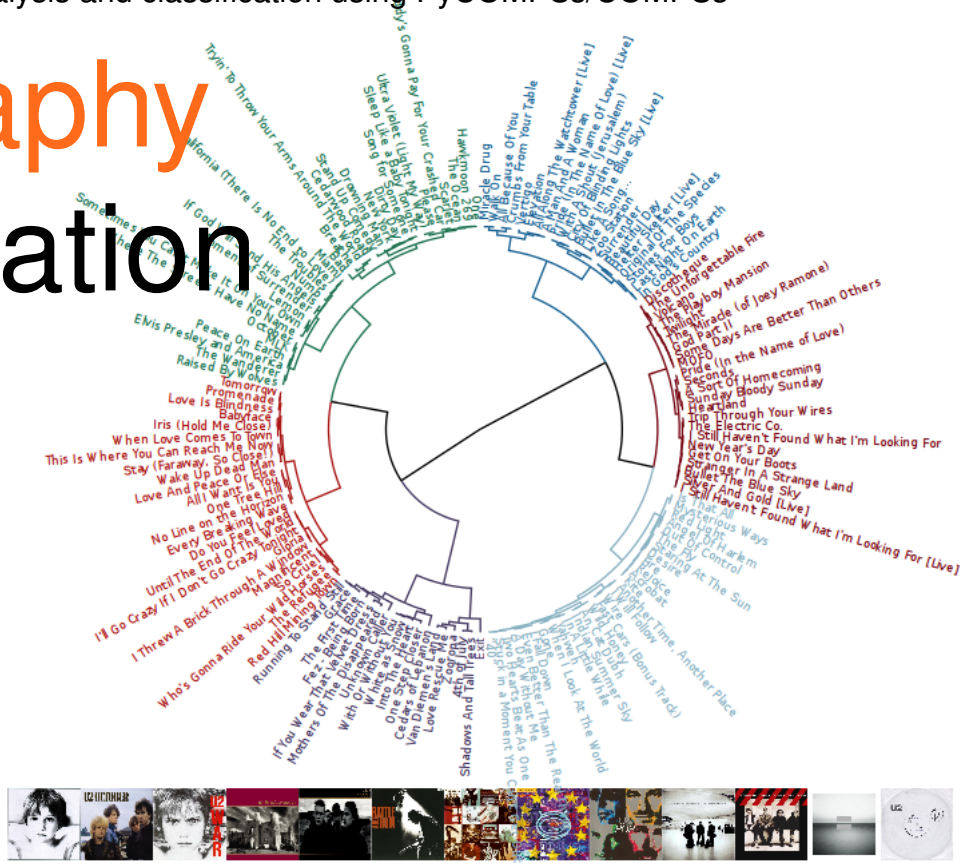
PRACE SoHPC2016 Coordinator
Leon Kos, University of Ljubljana
Phone: +386 4771 436 E-mail: leon.kos@lecad.fs.uni-lj.si

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


Leon Kos

# Discography Classification

*Sofia Kypraiou*




Several scientific papers and methods have been proposed for classifying music genres. But what happens if we want to know more? Not just about a single genre, but about an artist? In this project we used some of these methods to group songs of a single artist and learn more about their career and influences.

## Introduction

The idea which inspired the project was the enthusiasm to find out more about an artist and their career. Our goal was to classify similar songs by an artist into groups and thus be able to see the variety of their music. In this way, if someone has a specific song they like from an artist, they can see similar works by them. Cool, isn't it?

What makes the project even more interesting is the combination and requirement of many scientific fields - from machine learning techniques, statistics, data mining methods and of course supercomputers.

As input, we used the discography of various artists - artists with long music careers (U2), variation in their music style (Pink Floyd) and with many active years in both their music career and style (David Bowie).

Using an audio extraction application, the music files were then processed to extract some values which will later be used for their analysis. Some of the variables that are extracted and used include the bandwidth, the frequency and the pitch.

Not all of this data is needed though. The data needs to be cleaned, processed, and analysed to keep only their most useful features.

After the analysis of data, it is ready to be categorised and this is most optimally done using different clustering methods. This result was used for the final visualisation.

Due to the large dataset, parallelism is of course useful. The Barcelona Supercomputing Center (BSC) has developed COMPSs, a programming model and runtime that aims to parallelise sequentially programmed applications written in languages such as Python. It easily allows for parallelism to be achieved - even for those who do not have a strong programming background.

## Methods

As mentioned earlier, processing of music files goes through a number of stages, as described in the following figure:

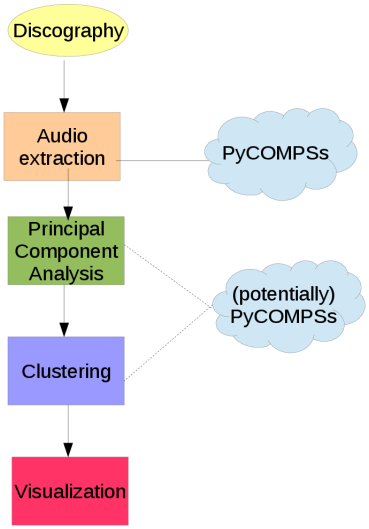



**Figure 1:** The stages until the final result

### Audio extraction

Audio extraction is used to extract audio feature sets which are later evaluated on their ability to differentiate. The feature sets include low-level signal properties - which refer to a physical descriptions of a song and mel-frequency spectral coefficients (MFCC) - which are commonly used in voice recognition and is based on human hearing perceptions. At the end of the audio extraction process, each song is described by 72 features.

### Data Analysis

Not all of these features are useful to the final result. Principal Component Analysis (PCA) is a statistical procedure used to emphasise variation and bring out strong patterns in a dataset. This was the method used to identify useful features - which were mainly those whose value often changed.

Sometimes we found that just two features were useful features - as shown in Figure 2.

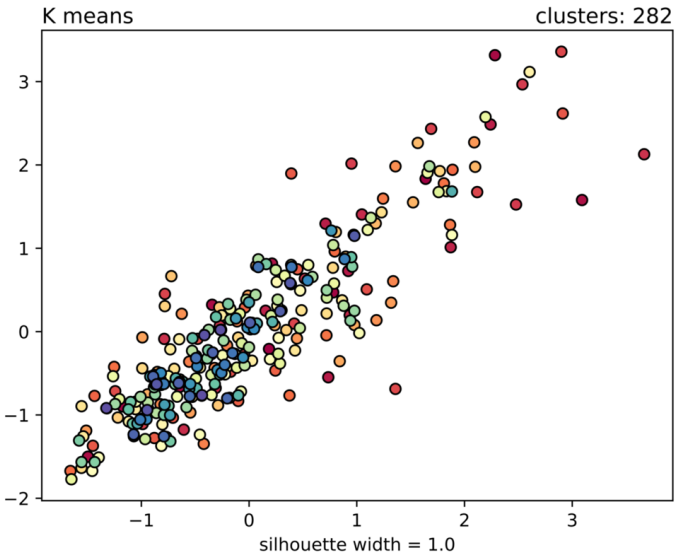



**Figure 2:** Clustering of David Bowie's discography using K means (2 features after the PCA).

### Clustering

Clustering analysis follows the cleaning of data and this combines data mining and machine learning to group together similar songs. Songs in the same group - called a cluster, are more similar to each other than songs in other clusters. Different clustering algorithms can be used for this - with each one producing different but comparable results.

But we don't know how many clusters there should be! How do we know which is the best number of clusters? Science is here to save the day, thanks to a metric called "silhouette width" that indicates how well the clusters are formed. Silhouette width has a range [-1,1], and a value close to 1 means that items within each cluster are alike, and that the groups are well separated. The following table shows different clusters created for Pink Floyd discography using different clustering algorithms and silhouette width.

**Table 1:** Clustering methods for Pink Floyd

| Clustering Method | Number of clusters | Silhouette width |
|---|---|---|
| Hierarchical | 4 | 0.0927 |
| Spectral | 57 | 0.1975 |
| KMeans | 89 | 0.6208 |
| DBSACN | 2 | 0.4161 |
| Propagation | 116 | 1.0 |

### Visualisation

Despite all this clever mathematics, clustering methods lack intuition and human inspection is necessary in the formation and determination of clusters. This is needed to gain an understanding of what the data represents and what the cluster represents and intends to achieve.

Data is better understood through visualising it and our data is brought to life using HTML, CSS and javascript library D3 web technologies to produce dynamic, interactive data visualisations in web browsers. The visualisation is the result of the cluster analysis combined with the user being able to define their own clusters.

### PyCOMPSs

For the purpose of performance, as well as benefiting from the potential of BSC's MareNostrum supercomputer, we used PyCOMPSs to parallelize the audio extraction stage. During this process songs are separated in chunks and each one of them is assigned to a node in the supercomputer. Below is the graph of the execution:

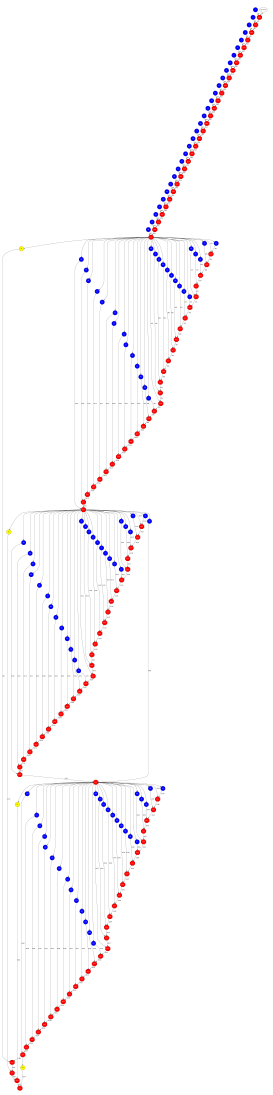


**Figure 3:** Graph of the execution. The blue nodes indicate the tasks whilst the red areas demonstrate where synchronisation was needed

## Results

It is interesting to see which features were useful. From the PCA data analysis, keeping 99% of the useful variables, we saw that only the MFCC features were contributing to the calculations.

Our first attempt with clustering methods did not turn out as expected. Figure 2 shows how clusters are merged

making it hard to distinguish them.
From Table 1, the silhouette width scores were low. A value lower than 0.3 means there is no structure in the data, while a value between 0.3 and 0.5 means there might be some structure.
Another interesting result was that it was almost impossible to cluster Pink Floyd's discography. Graph 4 shows that although we were able to fairly easily group U2 songs into 3 categories, for Pink Floyd the results were very poor.

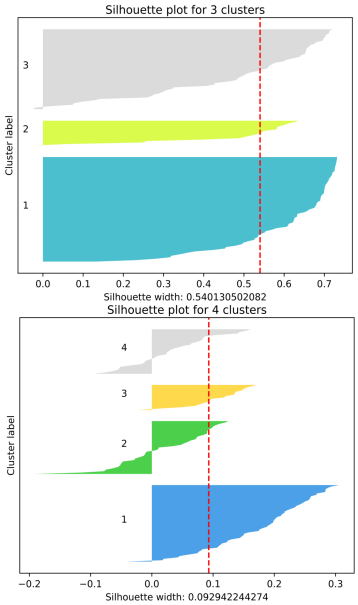



**Figure 4:** Best number of clusters for U2 (top) and Pink Floyd (bottom). Values below 0 indicate that the songs are wrongly classified.

Since that method did not work, samples were taken at 0, 30, 60, 90, 120 seconds of each song. After running hierarchical clustering on this data, all the introductions were put in one group, an indication that our clusters worked well. This is demonstrated in Figure 5 for U2 discography.

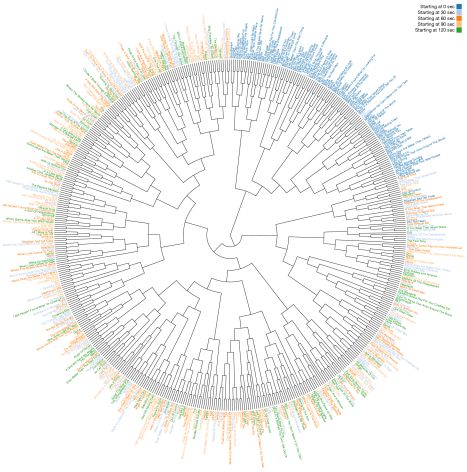


**Figure 5:** U2 discography with 5 samples per song. With the blue areas identifying introductions

For the final visualisation, we used the hierarchical clustering and specifically the dendrogram that it offers. For better optical results, we used a polar dendrogram as shown in Figure 6 and Figure 7.

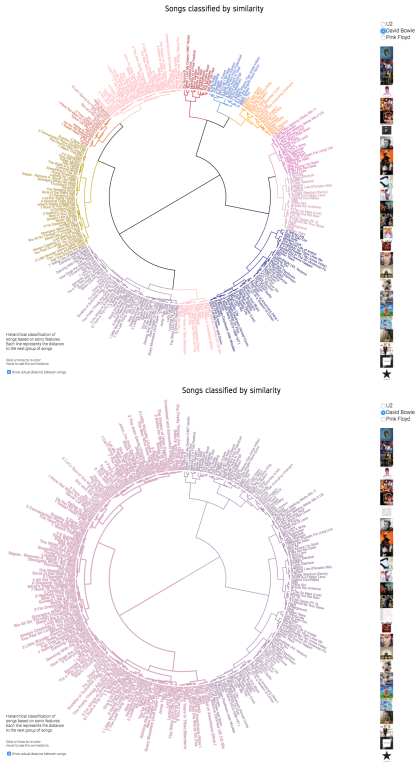



**Figure 6:** David Bowie discography. Top: default colouring of the clusters Bottom: user-defined clusters

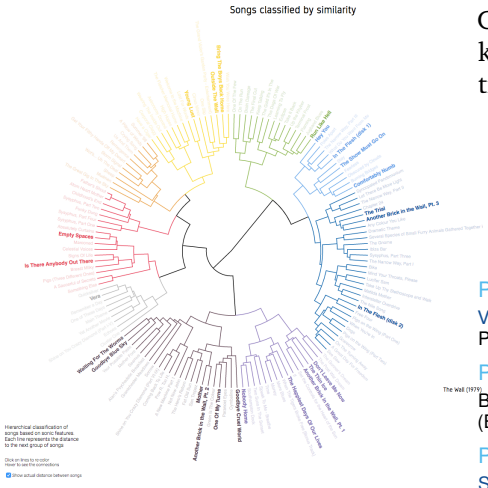



**Figure 7:** Pink Floyd discography. Viewing songs of the 'Wall' (1979)

The interactive dendrogam, along with the discographies of U2, David Bowie and Pink Floyd, can be found here.

## Conclusion and further work

All the described methods were used to discover more about the artist's music career. The fact that we did not get the expected results from the clustering means that MFCC features, although suited for genre classification, are not suitable for clustering songs of the same artists in the same genre.
An extension of our work can be the recommendation of similar songs of the same artist. This can be used in a variety of applications, such as recommendation of top-N songs of an artist, or in virtual DJ programs.

## References


[1] Jeroen Breebaart, Martin McKinne *Features for Audio Classification* 2003.

[2] Lindasalwa Muda, Mumtaj Begam and I. Elamvazuth *Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques* 2010


## Acknowledgements


I would like to thank my mentor, Fernando Cucchietti, for his guidance throughout the project and for providing the visualisations along with Guillermo Martin, the artist and Diana Fernanda Velez García for the the graphic design. Also Rosa Badia and Daniele Lezzi for their support with PyCOMPSs and last but not least, Carlos Carrasco Jimenez for the project and knowledge in machine learning, statistics, data analysis and data mining.


PRACE SoHPCProject Title
Visualization data pipeline in PyCOMPSs/COMPSs

PRACE SoHPCSite
Barcelona Supercomputing Center (BCS-CNS), Spain


PRACE SoHPCAuthors
Sofia Kypraiou, [National and Kapodistrian University of Athens,] Greece

PRACE SoHPCMentor
Fernando Cucchietti, BSC, Spain



Sofia Kypraiou


PRACE SoHPCContact
Fernando, Cucchietti, BSC
E-mail: fernando.cucchietti@bsc.es


PRACE SoHPCSoftware applied
Python, COMPSs/PyCOMPSs, D3.js

PRACE SoHPCMore Information
Music project COMPSs/PyCOMPSs, D3 JavaScript library

PRACE SoHPCProject ID
1601

Identifying visual features of castles with PyCOMPSs, through computer vision and machine learning.

# Computer Vision for SoHPC

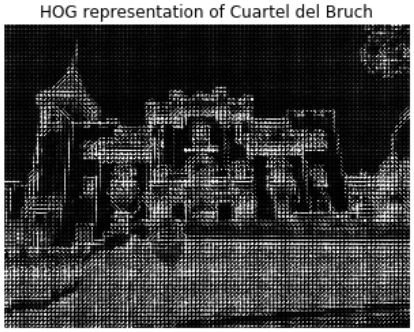
*Marco Forte*

Example detection of common castle features

This project will leverage the PyCOMPSs programming model and the computational power of the MareNostrum supercomputer to automatically classify images of castles and forts.

The focus of the project is on automatically extracting and identifying visual features of castles. Such features include towers, parapets and large stone walls. The result of the project is a geospatial and chronological map of castle features.

HOG representation of Cuartel del Bruch



ming model in an application which uses aspects of big data, computer vision and machine learning. PyCOMPSs is a programming model which allows a programmer to annotate their sequential code. In turn, PyCOMPS parallelises the code, taking care of the program flow and data dependencies automatically.

I was given the opportunity to design and implement the solution to this project myself. Computer vision was used to convert high dimensional data into a numeric form so that an algorithm can better distinguish contents of an image. This data was later used as input for machine learning. Many images had to be processed and thankfully this could be done in parallel - making it an ideal application for using HPC.

## Data retrieval

Close to 400 images of castles were downloaded from the Internet. These images were then broken down into image sections - called patches, of different shapes, sizes and scale. This resulted in about 100 patches per image.

## Approach

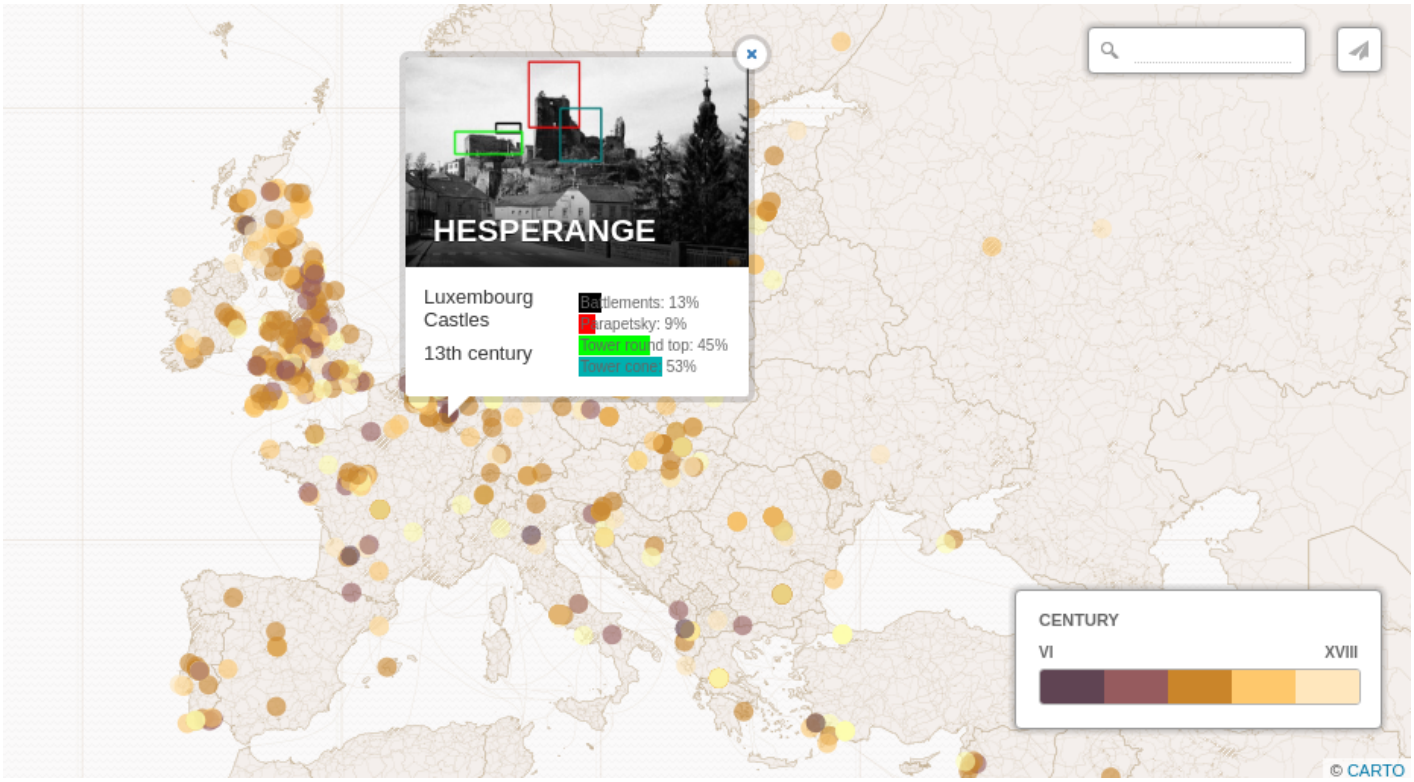
From all the resulting patches, those of high contrast were ignored - as these were mostly of earth and sky. For the remaining patches feature vectors were computed so as to describe the shape and colour in each patch.

Thankfully the above tasks can be carried out in parallel and using PyCOMPSs upon BSC's MareNostrum supercomputer a significant speed up in computation time was achieved.

### Discovering visual features unique to castles

The K-Means clustering algorithm can be applied to the remaining patches to identify clusters of common elements they share among them. After attempting this, it was found that the outcome would create clusters of commonly occurring but visually uninteresting corners and edges which are not necessarily castle like features.

In an effort to find visual elements unique to castles, a dataset of 10,00 images of other buildings which were not castles was also used considered. The idea behind this was that castle like fea-




The interactive map visualisation

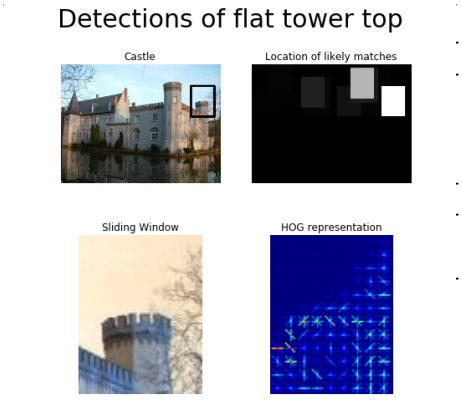
tures will appear more often in the castle image dataset than in the non-castle image dataset.

Upon the castle image dataset we found the most similar patches using the nearest neighbour classification algorithm. 1000 candidate patches were then selected by identifying those closest to 20 of their neighbours. In this way, unique castle like features could be identified better.

The "discriminative learning" approach was later used. Both the castle and non-castle image datasets were split into 3 parts. For each of the candidate patches a Support vector machine (SVM) classifier was trained using the castle image dataset nearest neighbours of the candidate patch as positive samples and all patches from the non-castle image dataset as negative samples. This was applied to the classifier to the castle image dataset and the top 5 positive matches were used for the next round.

The idea behind the above is that this will produce an increasingly accurate detector that is more discriminative towards castles. This SVM training step can be done in parallel for each patch using PyCOMPSs, with more fine grained parallelism for the application of the classifier using Python's MultiProcessing library.

Each resulting detector can then be tested in parallel on both image datasets and the most accurate of detectors can be identified. The dataflow from the pre-




work, they were able to automatically extract Parisian features from a large streetview dataset with better visual results - such as extracting the common window styles. The Paris paper achieved better results because features are much more common in each image, they vary less from image to image, have similar lighting, scales and orientations towards the camera.

The parameters of our algorithm were tweaked many times but due to lack of time it was decided use semi-supervised learning.

Multiple features of castles like battlements and towers were manually outlined and from there a fast template matching algorithm was used on the dataset to find many more examples. The template matching was set to a threshold which would allow for a few false positives which were later manually identified as hard negatives.

A SVM was then trained on each feature patch using feature matches marked as positive castle features and on negative non-castle features. The image on the top left show the manually selected image and the others show the positive matches automatically found.

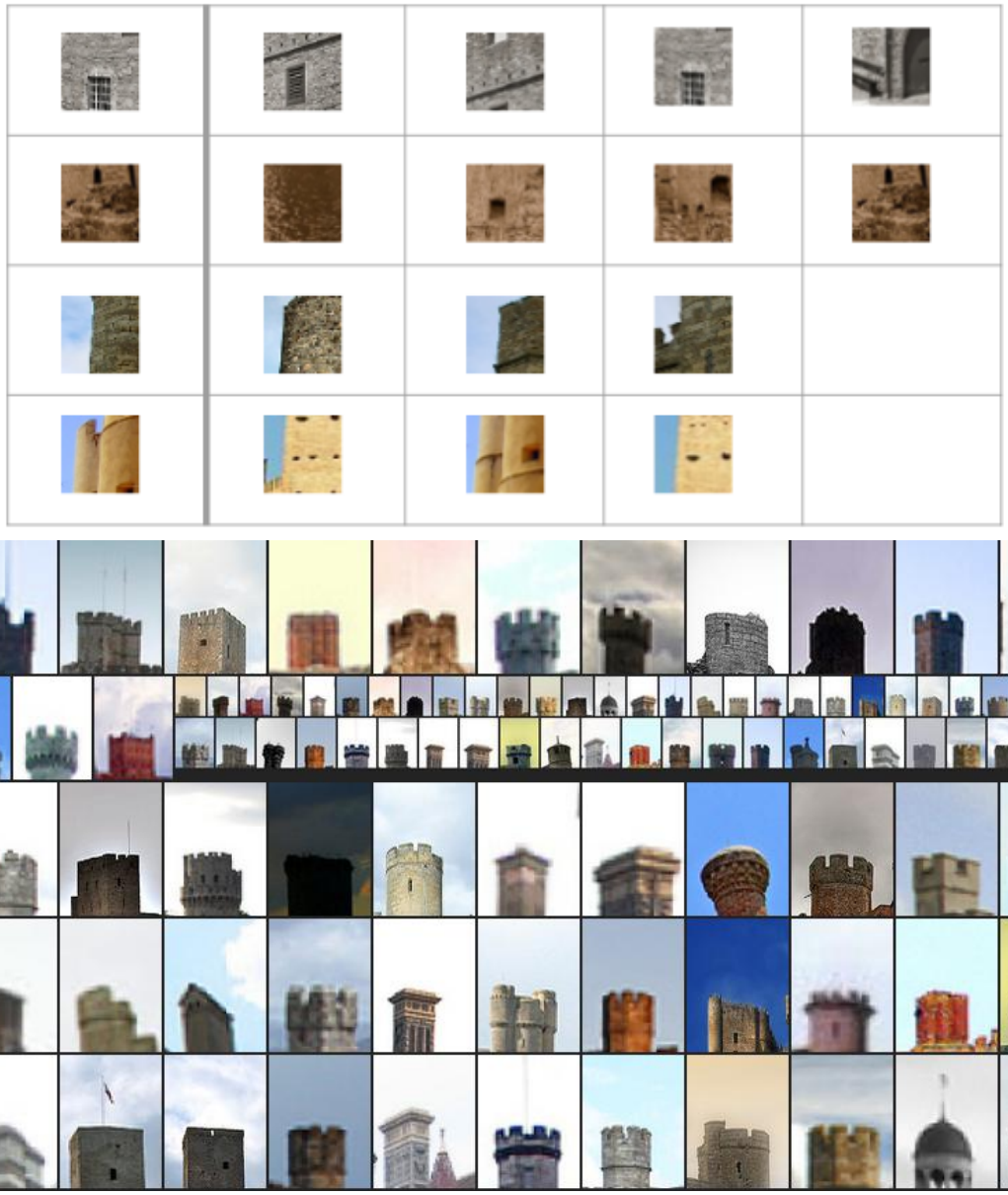
## Application of results

The final application of this was to a collection of 10,000 castle images which were sorted by castle name. This allowed for the retrieval of their construction date and coordinates from the Internet. For each castle we considered, each of our manually found features found the most probable match among the castles photos - with the probability of it being a match also given.

## Visualisation

This information is visualised as an interactive map of the castle locations across Europe. The map is available on the online mapping service CARTO at https://goo.gl/JXYsWU. For each castle a mosaic of images is shown with matches of the various features and the probability of a match given.

## Discussion & Conclusion



Automatically extracted patches and those semi-automatically found

Due to time constraints, the project is still not finished and improvements could still be made. The data sets that were used were relatively small compared to larger available datasets. A dataset of all castles in Europe as listed on Wikipedia is in the process of being collected and this will be much larger than the original.

But to be able to process such a dataset, code improvements will need to take place - such as selecting more appropriate libraries for computing HOG vectors and improving the PyCOMPSs parallelisation.

The project acts a good example of how HPC can be used in the context of machine learning, but also how HPC can be used in the field of humanities. HPC with machine learning allows for some work to automatically be done with computers rather than requiring thousands of trained expert hours and I believe there are many opportunities for HPC in the humanities such as in the fields like archaeology, history, art and architecture which could be exploited.

## References


Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A. Efros. What Makes Paris Look like Paris? ACM Transactions on Graphics (SIGGRAPH 2012), August 2012, vol. 31, No. 3.

Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). Vol. 1. IEEE, 2005.

Tejedor, Enric, et al. "PyCOMPSs: Parallel computational workflows in Python." International Journal of High Performance Computing Applications (2015): 1094342015594678.


## Acknowledgements


I would like to thank my project mentors Rosa M Badia and Daniele Lezzi for their support in my use of PyCOMPSs. I would also like to thank Fernando Cucchietti and Guillermo Marin for their assistance with the visualisation aspect of my project. Finally I would like to thank Carl Doersch for his inspirational "What Makes Paris Look Like Paris" paper and for the encouragement he gave me over email.


PRACE SoHPCProject Title

Development of sample application in PyCOMPSs/COMPSs

PRACE SoHPCSite

Barcelona Super Computing Center, Spain


PRACE SoHPCAuthors

Marco Forte, [Trinity College Dublin,] Ireland

PRACE SoHPCMentor

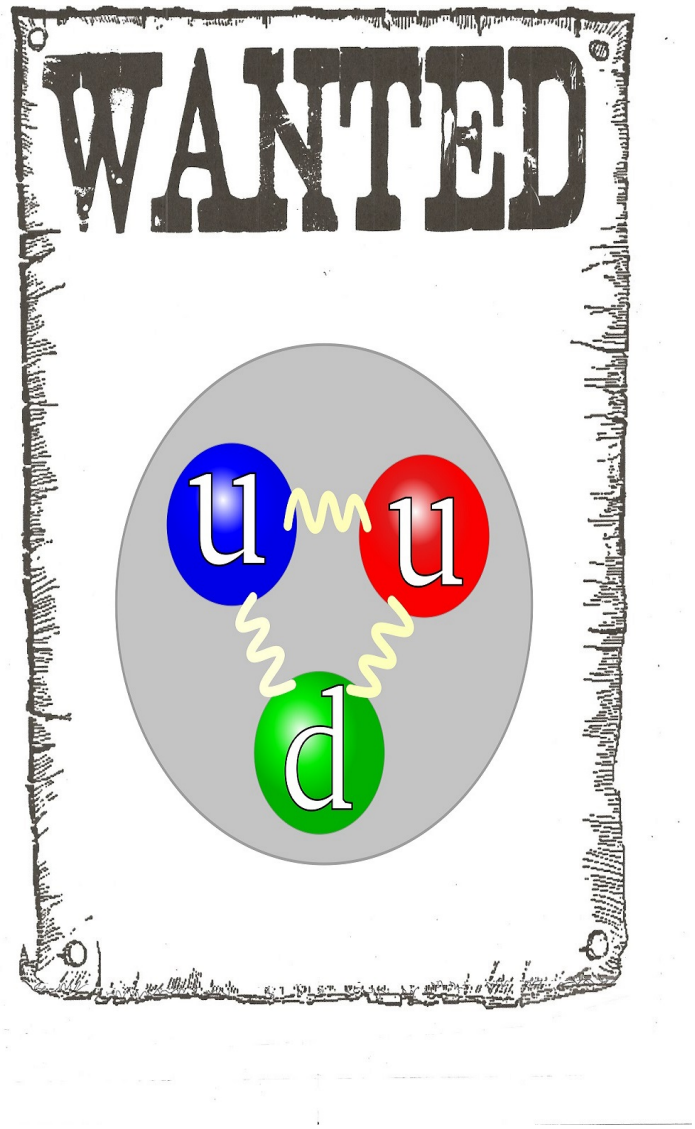
Rosa Badia, BSC, Spain


Marco Forte

An Investigation of the Positive and Negative Parity Excited States of the Nucleon in Lattice QCD at the Physical Point.

# Searching for Nucleon Excited States

*Shaun Lahert*




We investigate the low lying excited states of the nucleon in the positive and negative parity channels using the Generalised Eigenvalue Problem (GEVP). Athens Model Independent Analysis Scheme (AMIAS) is also preformed on the correlation function simulation data and we compare the results to the GEVP. Although the ground state of the proton is found, we highlight the problems of this approach in determining the exact spectra of the excited states.

Quantum chromodynamics (QCD) is the theory of the strong interaction from the standard model of particle physics. It describes particles which carry the colour charge - the 6 flavours of quarks and the gluons which transmit the interaction. In principle, any particle composed of these constituents - such as the proton, can also be accurately described by the theory.

Using lattice QCD - a reformulated version of the theory whereby we limit ourselves to a finite sized and discretised space-time, we can readily simulate the existence of such particles and their excited states. The aim of this project was to simulate the nucleon's excited states which have spin $\frac{1}{2}$ and which represent short lived particles with the same total isospin as the proton or neutron but which have higher energies. Isospin is a number related to the amount of up and down quarks in a particle.

## Euclidean Correlation Functions

The main property calculated on the lattice is the euclidean correlation function $\langle , \rangle$ of nucleon creation $\hat{\mathcal{N}}^{\dagger}(\mathbf{x})$ and annihilation $\hat{\mathcal{N}}(\mathbf{x}, t)$ operators. These operators have the same total isospin and spin as the proton and neutron and hence are composed of either two up quarks and 1 down quark operators for the proton or two down and one up for the neutron in a manner which gives spin $\frac{1}{2}$.

The physical interpretation of these correlation functions is that they give the probability amplitude of creating a nucleon at one point on the lattice and destroying it at another. With this modelling, we can compute the probability amplitude of a particle with the correct quantum numbers existing on the lattice with this amplitude being dependent on all the excited states with an effect related to their energies.

$$\langle \hat{\mathcal{N}}_i(\mathbf{x}, t) \hat{\mathcal{N}}_j^{\dagger}(\mathbf{x}, 0) \rangle^{\pm} = \sum_{n=0}^{\infty} A_i^{n\pm} A_j^{n\pm} e^{-E_n^{\pm} t} \qquad (1)$$

where $\pm$ refers to the positive and negative parity states and $i$ and $j$ represent index nucleon operators with different spin combinations of the up and down quark operators. The correlation function is computed on the lattice using the path integral formulation and the goal is to fit this sum of exponentials so we

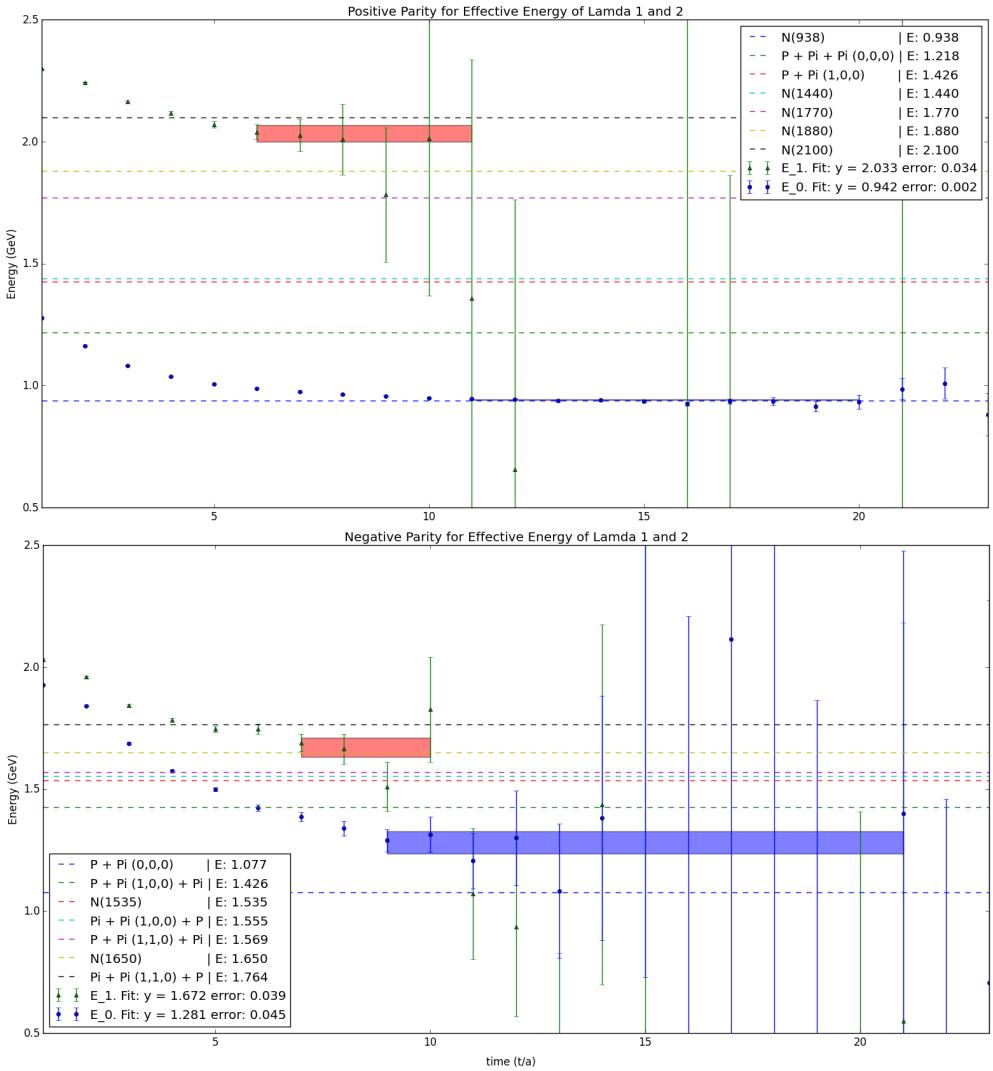



Fig 1. GEVP energy plateau fits for the positive parity (top) and negative parity (bottom) channels. The proton is easily identifiable at $0.940$ GeV in the positive parity channel For the negative parity channel the ground state is undetermined, however the first excited state at $1.65$ GeV seems to be resolved.

can extract the energies $E_n$ and hence the mass spectrum.

## Simulation Details

The correlation functions were generated from a periodic lattice with $N_f = 2$ twisted mass fermions, meaning the up and down quarks masses are equal. This results in the proton and the neutron having the same total mass. The table below contains the specific simulation meta-data.

| | |
|---|---|
| Pion mass ($m_\pi$) | 130 MeV |
| Lattice Volume | $48^3 \times 96$ |
| Lattice Spacing (a) | 0.093 fm |
| No. of Configurations | 2153 |

**Table 1:** Simulation Details

The pion mass being $135$ MeV is special as this corresponds to its real physical value - we call this being at the "physical point. This is new in lattice QCD simulations as a smaller pion mass means more computation. The number of configurations (samples) is also very high - so as to hopefully increase accuracy in the results. The lattice spacing was determined by the value of the pion decay constant. Two different nucleon operators were chosen to generate a $2 \times 2$ correlation matrix.

$$\hat{\mathcal{N}}_1 = \left(\hat{u}^T C\gamma_5 \hat{d}\right)\hat{u}$$
$$\hat{\mathcal{N}}_2 = \left(\hat{u}^T C\hat{d}\right)\gamma_5\hat{u}$$

## Genealised Eigenvalue Problem (GEVP)

We have a $2 \times 2$ matrix of the form

$$\mathbf{C}^{\pm}(t) = \sum_{n=0}^{\infty} A_i^{n\pm} A_j^{n\pm} e^{-E_n^{\pm} t} \qquad (2)$$

The GEVP is then defined as

$$\mathbf{C}(t)\vec{v}_n = \lambda_n(t)\mathbf{C}(0)\vec{v}_n \qquad (3)$$

with

$$\lambda_n(t) = e^{E_n t} \qquad (4)$$

We can then extract the energies of the low lying states from the eigenvalues by a plateau fit of

$$\log\left(\frac{\lambda_n(t)}{\lambda_n(t+1)}\right) \qquad (5)$$

These plateau fits can be seen above in Figure 1 for our two eigenvalues for both the positive and negative parity channels. The ground state of the proton is easily seen in the positive parity channel at $0.940$ GeV. The first excited state is seen at $2.03$ GeV which does not coincide with any known excited state. Despite this, the data is quite noisy at this point and it is not obvious where the plataeu begins.

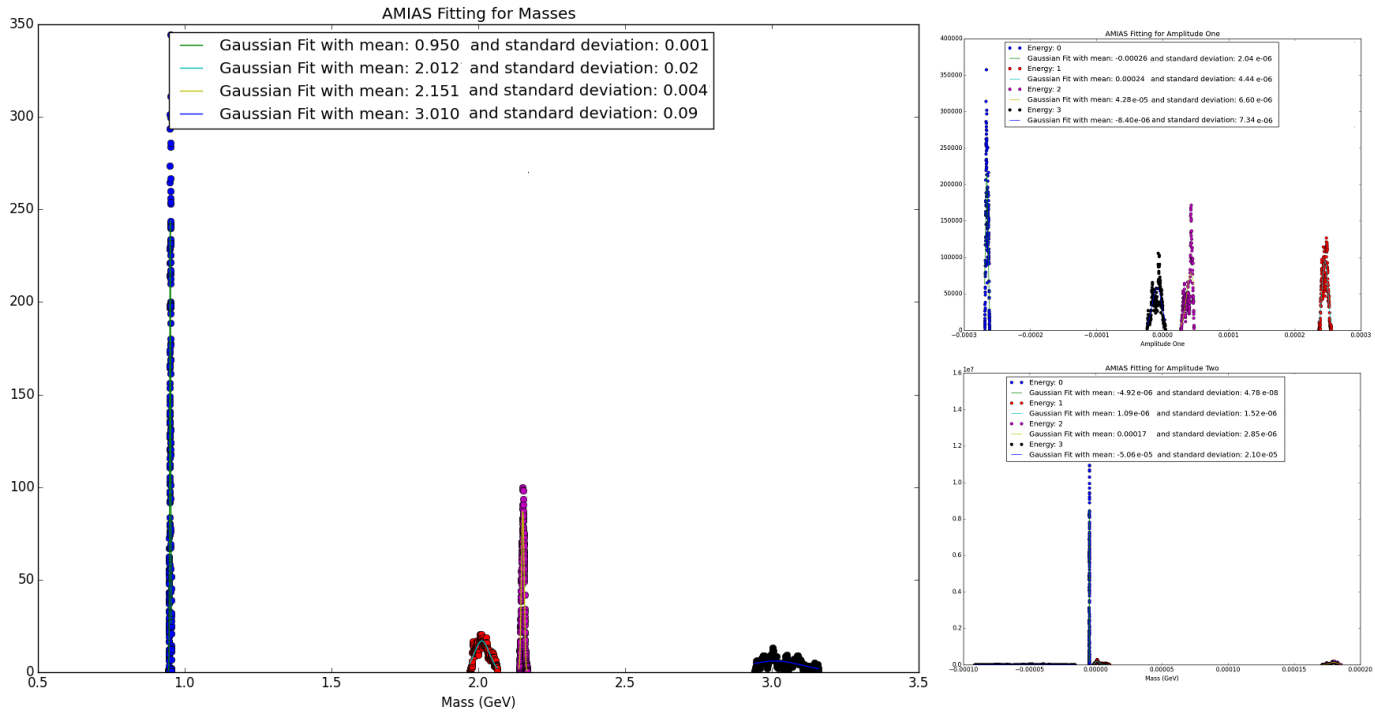



Fig 1. GEVP energy plateau fits for the positive parity (top) and negative parity (bottom) channels. The proton is easily identifiable at $0.940$ GeV in the positive parity channel For the negative parity channel the ground state is undetermined, however the first excited state at $1.65$ GeV seems to be resolved.

For the negative parity channel the ground state is also undetermined, however the first excited state appears to coincide with the expected first excited state of $1.650$ GeV.

## Athens Model Independent Analysis Scheme (AMIAS)

AMIAS is an alternative approach to fitting correlation function data, whereby each set of fit parameters $\{A_i^n, E_n\}$ is given a weight by the value of $e^{-\frac{\chi^2}{2}}$ fit function:

$$\sum_{ij,t} \frac{\left(\langle C(t)_{ij}\rangle - \sum_{n=0}^{\infty} A_i^n A_j^n e^{-E_n t}\right)^2}{\left(\sigma^{ij}(t)/\sqrt{N}\right)^2} \tag{6}$$

where $N$ is the number of configurations and $\langle C(t)_{ij}\rangle$, $\sigma^{ij}(t)$ are the mean and standard deviation of the data.
Using Monte Carlo techniques to sample this 'parameter space' we can generate probability distributions for each individual fit parameter according to the weight. These distributions will be gaussian for large enough sample sizes and hence we can extract the best fit parameters and the errors from the mean and standard deviation of the gaussians.

Figure 2 contains the best fit parameters for the first four energy levels of the positive parity channel. Again we see the proton clearly resolved at $0.950$ GeV with the determined excited states at $2.012$, $2.151$ and $3.010$ GeV. We have a strong agreement between these values and the values from GEVP method. We can also see from our second amplitude plot that the second nucleon interpolating field has a very little overlap with the excited states.

## Conclusion

Although the proton clearly showed up in the data, the other excited states, specifically the first positive parity excited state and the negative parity ground state did not appear. This is disappointing as the simulation was carried out at the physical point with a large number of configurations. Alternative simulation approaches must be considered to properly resolve these missing single particle excited states. Also, given that no obvious multi-particle states appeared in the data, consideration should be given to the use of multi-particle interpolating fields for the nucleon + pion.

## References


[1] C. Alexandrou, T. Korzec, G. Koutsou, T. Leontiou (2014) Nucleon Excited States in Nf=2 lattice QCD *arXiv:*1302.4410

[2] Blossier, B. Della Morte, M. von Hippel, G. Mendes, T. Sommer, R. (2009). On the generalized eigenvalue method for energies and matrix elements in lattice field theory. *DOI* 10.1088/1126-6708/2009/04/094.

[3] Papanicolas1, C.N. Stiliaris, E. (2012). A novel method of data analysis for hadronic physics *arXiv:*1205.6505

[4] K.A. Olive et al. (Particle Data Group) Chin. Phys. C, 38, 090001 (2014) and 2015 update.


### PRACE SoHPCProject Title

Searching for Nucleon Excited States at the Physical Point in Lattice QCD.

### PRACE SoHPCSite

The Cyprus Institute, Cyprus

### PRACE SoHPCAuthors


Shaun Lahert, [Trinity College Dublin,] Ireland


### PRACE SoHPCMentor

Constantia Alexandrou, The Cyprus Institute, Cyprus

Shaun Lahert

### PRACE SoHPCContact


Shaun, Lahert, Trinity College Dublin
Phone: +353851572847
E-mail: laherts@tcd.ie


### PRACE SoHPCAcknowledgement

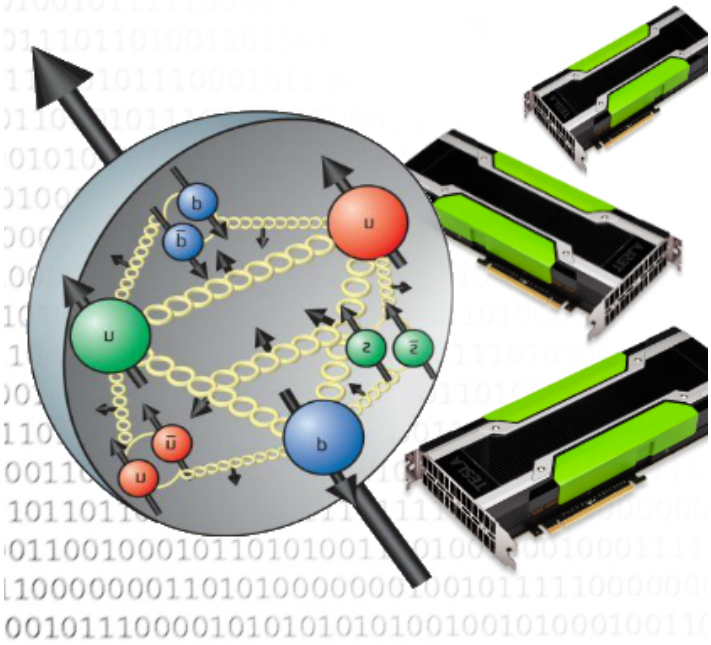

I would like to thank PRACE for giving me the oppurtunity to work on this project. I would like to thank my mentors Constantia Alexandrou and Giannis Koutsou for their time and effort in helping me. Finally I would like to Jacob Finkenrath for dedicating a sizeable part of his day making sure we knew what we were doing.


### PRACE SoHPCProject ID

1603

# Multigrid: It's time to speed it up!

*Ambra Abdullahi Hassan*



The tiniest particles from Quantum ChromoDynamic require the greatest computational resources. We tackle this problem by speeding up the most computationally expensive process of Lattice QCD simulations - the linear solver. Combining hardware (GPUs) and software (Multigrid) we will achieve to go faster.

There is a theory that confines quarks and gluons into neutrons and protons and this explains how it is possible for all the mass in our universe to exist. This fascinating theory discloses the deepest secrets of physics - from how the sun burns to the state of the universe immediately after the big bang.

Lattice QCD discloses the secrets of our universe.

This theory is called Quantum Chromodynamics (QCD). Its equivalent 'computer-friendly' counterpart, called Lattice QCD is one of the biggest challenges a computer scientist working with High Performance Computing (HPC) can face. Lattice QCD is capable of exploiting HPC to its maximum. Scientists working on Lattice QCD develop new algorithms for massively parallel systems and a great number of computer architectures have been created and designed specifically to solve Lattice QCD problems. It is not by chance that Lattice QCD was in the original 'Grand Challenges' and why so many Summer of HPC projects are related to Lattice QCD. Indeed, about a quarter of PRACE allocated resources are for Lattice QCD related projects!

In Lattice QCD simulations, most of the computational time is spent solving (or in other words inverting) one or more linear system of equations. Solving a linear system of equations is something that we all usually learn to do at high school - how many times were you asked to find the values of the variables X and Y in the following figure?

$$X + 2Y = 4$$

$$X - Y = 1$$

Of course, in Lattice QCD linear systems are huge, comprised of millions of variables. Solving a linear systems is crucial and to obtain better performance, the time needed for this operation needs to decrease.

## GPUs and multigrid

One of two ways can be used to obtain better performance:

- Use different hardware. For example, use a more powerful supercomputer, or increase the number of processors.
- Use different software. For example, using a more efficient method or algorithm.

In this project we sought to exploit both of the above. We chose to use Multigrid Methods - that have proved to be very efficient, and we wanted to do this using Graphic Processor Units (GPUs).

**Figure 1:** NVIDIA Tesla K80, one of the GPUs that help us boosting Lattice QCD performances

## The hardware: JURECA GPUs nodes

GPUs were originally created for manipulating computer graphics and 3D images. GPUs have an intrinsic parallel structure, with each GPU possessing hundreds of cores that can handle thousands of threads. GPUs can now be found everywhere - in laptops, in embedded systems, in video game consoles and in mobile phones.

It did not take long for HPC scientists to start considering using GPUs parallelism for scientific computing as well. Nowadays, GPUs are used as coprocessors (along with normal CPUs) in scientific computations. We too used GPUs in our project to accelerate our linear solver.

Most of the results obtained during the project have been obtained using the JURECA cluster which has 128 GPU-accelerated nodes with each node having two NVIDIA Tesla K80 GPU - each with a dual-GPU design. We were able to run our application using CUDA upon these nodes. We thus had available up to 4 GPUs per node and up to 512 GPUs available to us.

## The software: QUDA and multigrid

QUDA is an Open Source library for solving Lattice QCD problems on CUDA. The project started in the 2008 at the Boston University and the library now provides an efficient framework for Lattice QCD on GPUs including mixed-precision methods, domain decomposition preconditioners and multigrid solvers.

In multigrid, the linear system is solved two or more times on grids of different sizes (also called coarse and fine grids or levels). After all the levels have been built, the approximate solution is projected from a fine to a coarse grid and viceversa.

## Tuning multigrid parameters and boosting performance

Multigrid methods require users to choose among a large variety of parameters. Consequently, in order to obtain good performance, tuning of parameters is necessary. Above all, two parameters have been shown to speed up the inversion time - block size and mu shift.

### Block size

Block size controls the reduction in size of the lattice when creating the coarse grid level. It has a great impact on performance - regardless of the number of levels and the configuration chosen, a big block size shortens inversion time.

**Table 1:** The total solution time decreases as the block size increases. Results are for a 16x16x16x32 lattice using a 2-level multrigrid preconditioner

| Block size | total time (sec) |
|---|---|
| 2x2x2x2 | 161 |
| 2x2x2x4 | 93 |
| 4x4x4x4 | 51 |

Unfortunately, block size can not be too big. The final lattice must have at least a size of 2 in all dimensions. Since the size of the final lattice depends on both the block size and the number of levels coupled with the number of GPUs we are using, using a big block size can be impossible when using many GPUs or many levels.

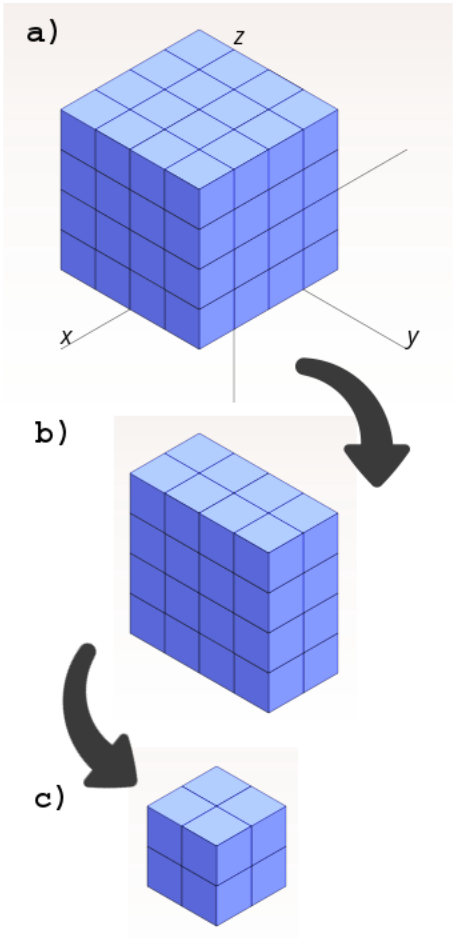
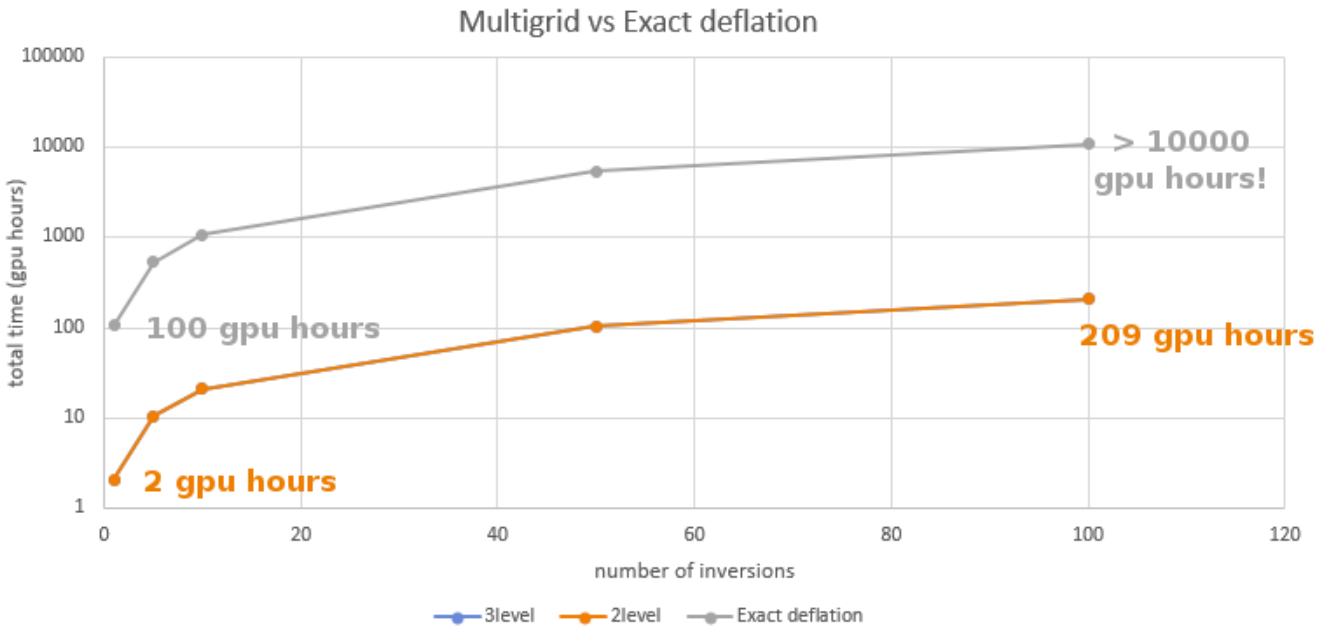



**Figure 2:** Global and local lattice size. a) the global lattice size is 4x4x4 b)2 GPUs are distributed along the z-axis. Thus, on each GPU the local lattice size is 4x4x2 c) When building the second level, we use a block size of 2x2x1 thus on the second level the local lattice size is 2x2x2

### Mu shift

Block size is a useful parameter to tune, but it has some severe limitations - the worst of all being that it affects scalability! Fortunately there is another parameter that was introduced at the University of Cyprus, which has been shown to greatly improve performance. This parameter is called mu shift and it has a magical power - if you apply the shift to the operator at the coarse levels, it becomes easier to invert it. Amazing, isn't it? This parameter really did the trick. Considering a physical lattice of dimension 48x48x48x96 and 24 GPUs to work with, it was very difficult to invert it. Every time we tried to apply the multrigrid, it failed miserably with one inversion requiring hours of computation. Not giving up, we decided to apply the shift and the inversion time decreased considerably. At the end, we




Our main result: After tuning our multigrid behaves better than the exact-deflation solver (note how much time we can save running 100 inversions!)

could perform an inversion in less than 20 seconds!

## 1 Comparisons

Thanks to the tuning on the block size and on the shift developed in Cyprus, we achieved a best time of 20 seconds per inversion. This is a pretty good result, since our first attempts needed several hours to invert the same linear system! After showning that the method scales with the number of GPUs, the final test for our multigrid method is the comparison with another very good method called exact deflation CG. In order to make a fair comparison, we had to reproduce the standard conditions in which Lattice QCD simulations are run. In most Lattice QCD applications, scientists almost never perform a single inversion - usually the same linear systems is inverted several times, changing only the right hand side vector (the rhs vector in short). For this reason, we wanted to compare the performance of the two solvers not for just a single inversion, but for many of them.

It's time to announce the winner! Our multigrid method performs better! As can be seen, it outperforms the exact deflation CG both when performing a single inversion (multigrid setup time is very short) and when performing multiple inversions (even the inversion time is shorter). You can see that our method behaves very well in the case of few inversions (this is because our linear solver has a much smaller setup time, compared to exact deflation GC), but it is very good even in the case of many inversions (this is because even the inversion time is slightly better than the standard solver). In the figure you can see how much time we can save in terms of GPU hours, such as the number of hours multiplied by the number of GPUs.

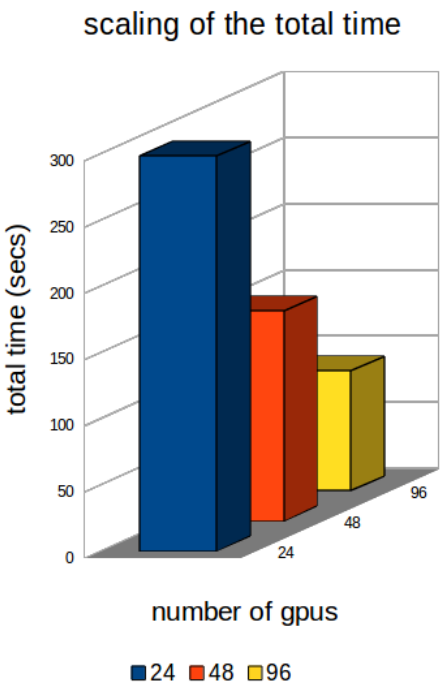



**Figure 3:** total time scales well with the number of GPUs


## Acknowledgements

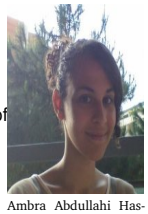
During the two months at the Cyprus Institute many people have helped me in successfully completing my project. Thanks to my project coordinator Giannis for the patience and the help he gave us during this summer. Thanks to Jacob, who constantly helped me during these 2 months. Thanks to Christos as well who wrote the modification of the code that allowed me to obtain these results and helped me greatly in understanding the QUDA multigrid. Thanks to our site coordinator, Stelios and thanks to all the people at the Cyprus Institute that made me feel at home.


PRACE SoHPCProject Title
Multigrid methods for Lattice QCD

PRACE SoHPCSite
Cyprus Institute, Cyprus


PRACE SoHPCAuthors
Ambra Abdullahi Hassan, University of Rome Tor Vergata, Italy

PRACE SoHPCMentor
Giannis Koutsou, Cyprus Institute, Cyprus



Ambra Abdullahi Hassan

PRACE SoHPCSoftware applied
QUDA

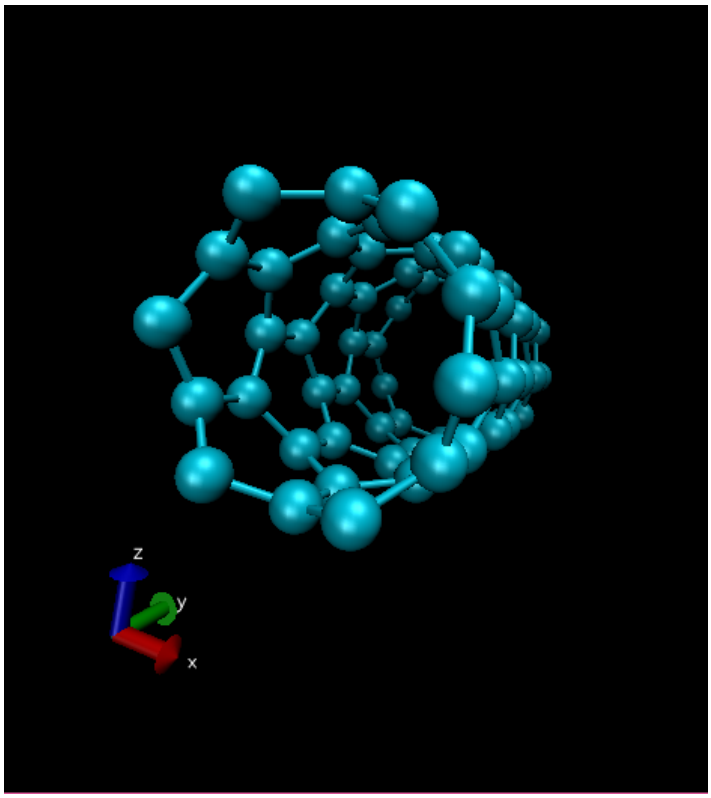
PRACE SoHPCMore Information
https://lattice.github.io/quda/

PRACE SoHPCProject ID
1604

Fortran, C and MPI perfectly combined to perform calculations of nanotubes by utilising the helical symmetry properties

# Calculating Nanotubes

*Katerina Galata*




Nowadays nanotubes are widely used both in research and industry, so their properties are already calculated - but only through methods based on 1D translational symmetry using a huge unit cell. In this project there is a pseudo 2D approach for fully ab initio calculations of nanotubes that explicitly uses the helical symmetry properties. The systems considered are huge and so the code needs to be parallel to be efficient.

Nowadays nanotubes are not only used in research, but there are also attempts to use them on a large scale in commercial products. Because of this, their properties need to be examined so the properties of the final products can be predicted. Given that experimental property testing costs more than supercomputing calculations, everybody is trying to achieve this through simulations. Calculations of nanotubes are usually performed using methods based on a one-dimensional symmetry and a huge unit cell. Meanwhile, a pseudo two-dimensional approach (which is the approach when the inherent helical symmetry of general chilarity nanotubes is exploited), has been tested only by utilising the simple approximate Hamiltonian models.

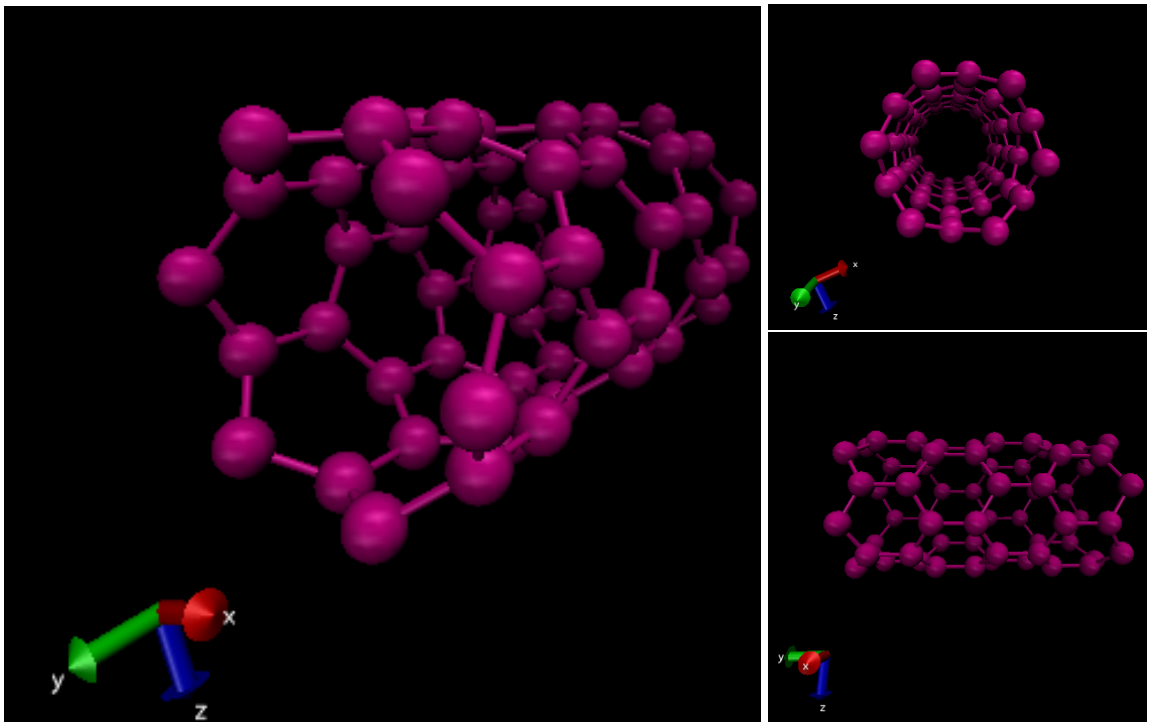
But this is not enough when high quality products are required. Given that nanotubes are used in dental implants, artificial muscles and even when somebody is trying to get rid of cancer, the calculations have to be precise. This is the reason why this code is developed. Through this code, fully ab-initio calculations of nanotubes are performed, explicitly using the helical symmetry properties. Implementation is based on a formulation in two-dimensional reciprocal space where one dimension is continuous and the second is discrete.

## Theoretical Background

Independent particle quantum chemistry methods, such as Hartree-Fock and/or Density Functional Theory or simple post Hartree-Fock MP2 are used to calculate the band structures.

For instance, the Hartree-Fock theory suggests that each individual electron moves independently of each other, only taking into consideration the average electrostatic field due to all other electrons and the field due to the atoms. According to this theory, exchanged interactions are forced by forcing the antisymmetricity of the electronic wavefunction. This acts to lower the total binding energy of atoms by ensuring that parallel spin electrons stay apart. The downside to the theory is that it neglects correlations in the motion between two electrons with anti parallel spins.

With regards to DFT, the functional (functional is a function within a function) is the electron density which is a function of space and time. The elec-



Results of the simulation that took place during the project. These nanotubes consist of Boron atoms.

tron density is used here as the fundamental property unlike the previous theory (H-F), which significantly speeds up the calculations (now there are only three variables -x,y,z- while before there were 3N variables). But still, the systems examined consist of thousands of atoms. So, as anybody can easily understand the calculations require a significant amount of time to complete. And that is why supercomputers and parallel programming are needed.

## Using MPI

### What is MPI?

MPI stands for Message Passing Interface. This is the tool used to make computer programs run faster on supercomputers. The main principle is that the workload is shared out as tasks, as equally as possible among different computational nodes. At the end, the outcomes of these tasks are combined to make up the final result.

### Hands on

The toughest part when somebody is trying to parallelise a code is writing the correct commands. Of course not! What makes things complicated are the previous coder/coders. Usually in such projects there is always a group of people working on a piece of code. So throughout the years, everybody is using a previous edition of the code, developing it slightly and this keeps going. Time is required to fully understand the implementation of previous coders. Once understood the fun can begin.

What was implemented in this project, was the parallelisation of routines that performed certain functions - such as the diagonalisation of a matrix. MPI can operate using a Master/Slave scheme. The master is always aware of everything that is going on within the code and the slaves are responsible of executing some specific tasks as noted by the user in the code. So how is this code structured? First of all, a routine that gives directions of what the slave needs to do is written. This routine calls the subroutine that needs to be parallelised so slaves will be able to go through the tasks that are assigned to them. But in order to do that, the user first has to make sure that all the variables are initialised and the work-arrays are allocated to both the master and the slaves. After that, in this project the master and the slaves go through some loops alternately, with the master always knowing the results of each iteration performed. As a result, there is an improvement on the performance of the code, but still a lot have to be done in the future, since there are a lot of other routines that need to be parallelised.

## References

[1] Charlotte Froese Fischer General Hartree-Fock program Computer Physics Communications Volume 43, Issue 3, February–March 1987, Pages 355-365

[1] Nathan Argaman, Guy Makov Density Functional Theory – an introduction American Journal of Physics 68 (2000), 69-79

### PRACE SoHPCProject Title

Calculation of nanotubes by utilizing the helical symmetry properties

### PRACE SoHPCSite

SAS-Slovak Academy of Sciences, Slovakia

### PRACE SoHPCAuthors

Katerina Galata, , NTUA-National Technical University of Athens, Department of Chemical Engineering, Greece


Katerina Galata

### PRACE SoHPCMentor

Prof. Dr. Jozef Noga, SAS, Slovakia

### PRACE SoHPCContact

Leon Kos
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

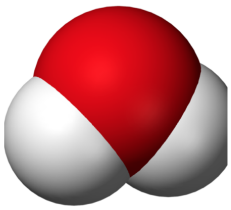
I would like to first of all, thank PRACE and the SoHPC organisers for giving me the opportunity to participate in a program with such great outcomes. Furthermore, I am grateful to my mentor, Prof. Jozef Noga, who let me take part in such an important project and gave me the necessary directions through the process. Last but not least I would like to express my gratitude to Lukáš Demovič, who was always there to answer every question I had (so I learnt a lot of new stuff) and who made my staying in Bratislava very pleasant with his hospitality.

### PRACE SoHPCProject ID

1605

# Quantum Chemistry in Spark for SoHPC

*Oisín Benson*

The goal of the project was to implement and test the performance of simple quantum chemistry methods, such as the Hartee-Fock method using Scala and Spark - a general-purpose engine for large-scale data processing. After successfully running a Scala prototype, a Spark implementation was prepared which was able to run for small basis sets. Unfortunately the code requires more refinement and still diverges for large basis sets.

The focus of this project was to implement and test the performance of the Hartee-Fock method from quantum chemistry using Apache Spark - a general-purpose engine for large-scale data processing (an alternative to Hadoop/MapReduce). Initially developed by Matei Zaharia at UC Berkeley's AMPLab in 2009, and open sourced in 2010, Spark has seen a relatively rapid adoption in the area of big data due to its speed when compared to Hadoop, supporting applications in multiple languages and its relative ease of use. Spark has even received interest from NASA for some of their large scale data processing needs. Spark's fault tolerance with node-aware distributed storage, caching and automated memory management could compensate for the loss of performance when compared to a pure HPC language such as MPI Fortran.

Spark runs on top of the Java Virtual Machine (JVM), thus cannot match the performance of Fortran/C(++) compiled programs, but it does have many desirable features of a distributed parallel application such as fault-tolerance, node-aware distributed storage, caching and automated memory management.

## Spark

The core concept of Spark is that of the Resilient Distributed Dataset (RDD).[1] This distributed memory abstraction allows programmers to perform in-

memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that many computing frameworks handle inefficiently - iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude.

To achieve fault tolerance efficiently, RDDs provide an interface based on coarse-grained transformations, such as map, filter and join, that apply the same operation to many data elements. For example, map is a transformation that passes each element of a dataset/RDD through a function and returns a new RDD representing the results. This allows them to efficiently provide fault tolerance by logging the transformations used to build a dataset rather than the actual data. The RDD itself contains all the dependency information needed to replicate each of its partitions. Thus, if a partition is lost, the RDD has enough information about its lineage to recompute it, which can be parallelised to allow for faster recovery. This ability to recover data via lineage also means that RDDs do not need to incur the overhead of checkpointing which requires a snapshot of the entire program.

Spark code can be written in various languages including Python and Java. We decided that the functional programming language Scala was best suited to the task. Efficiently written, Scala heavily utilises many of the coarse grained transformations (such as map and reduceByKey) common to Spark and was thus a natural fit for interfacing with Spark. Another advantage to writing the code in Scala was its numerical/linear algebra library Breeze which is also present in Spark through its machine learning MlLib library. This efficient and clean library uses netlib-java which loads Fortran/C linear algebra libraries via JNI.

## Hartree-Fock

Using these tools we attempted to implement the Hartree Fock method from quantum chemistry, primarily in Scala and then in Spark. Hartree-Fock theory is fundamental to much of electronic structure theory as it provides a method of approximation for determining the wave function and the energy of a quantum many-body system in a stationary state. It is the basis of molecular orbital (MO) theory, which posits that each electron's motion can be described by a single-particle function (orbital) which does not depend explicitly on the instantaneous motions of the other electrons. Hartree-Fock theory can only provide an exact solution in the case of the hydrogen atom, where orbitals exact eigenfunctions of the full electronic Hamiltonian. However, the Hartree-Fock theory often provides a good starting point for more elaborate theoretical methods which are better approximations to the Schrödinger equation (many-body perturbation theory).

The libint C library's[2] Hartree Fock method was rewritten in Scala to set up an initial prototype. One electron integrals are first read into the program which form a basis set allowing us to construct the core Hamiltonian.

$$H_{\mu\nu}^{core} = T_{\mu\nu}^{kinetic} + V_{\mu\nu}^{potential} \quad (1)$$

The overlap matrix was then orthogonalised and used to construct the initial Fock matrix. Its eigenvectors and the number of occupied molecular orbitals($(C_0)_{\mu}^{m}$) were then used to construct the initial Density matrix $D_{\mu\nu}$ as follows:

$$D_{\mu\nu}^{0} = \Sigma_{\mu\nu}^{occ}(C_0)_{\mu}^{m}(C_0)_{\mu}^{m} \quad (2)$$

All of these operations could be performed with breeze matrices. Computing the new Fock Matrix $F_{\mu\nu}$ for subsequent iterations required reading the 4-dimensional two electron integral. Using the integrals 8-fold symmetry and resulting degeneracy the algorithm updated the $(i,j),(k,l),(i,k),(j,l),(i,l)\&(j,k)$ of the $F_{\mu\nu}$ matrix using the previous $D_{\mu\nu}$ as below.

$$F_{\mu\nu}^{i} = H_{\mu\nu} + \Sigma_{\lambda\sigma}^{AO} D_{\mu\nu}^{i-1}[2*_{\mu\nu|\lambda\sigma} - _{\mu\lambda|\nu\sigma}] \quad (3)$$

Once this was done, a new density matrix was constructed from the updated $F_{\mu\nu}$ matrix and the energy could be computed as follows:

$$E_{elec}^{i} = \Sigma_{\mu\nu} D_{\mu\nu}(H_{\mu\nu} + F_{\mu\nu}) \quad (4)$$

The previous three steps were then repeated until the energy converged.

## Spark Prototype

The processing and data transformations of the two electron integral necessary to create the new $F$ matrix proved the central challenge of the project. To load the 4d Integral dataset as an RDD we first needed to convert large datasets from Binary→HDFS text file. This dataset was structured as i j; followed by the list of unique k, l values. Once it was stored in this format, the spark code could call an integral as RDD and preform map, split along with other data transformations to extract the i, j, k, l index and value. From here, the rest of the libint update was performed and the resulting RDD could be added to $F_{\mu\nu}^{new}$. This section of code was finished with Collect to force the evaluation of the RDD.

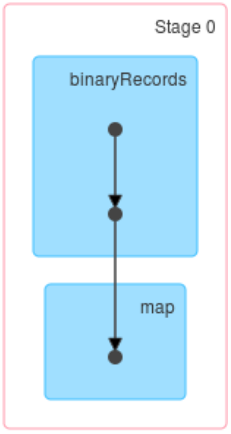



**Figure 1:** A simple Directed Acyclic Graph for reading or mapping a binary file to an RDD

Spark works by creating operator graphs called Directed Acyclic Graphs (DAG). The DAG scheduler divides operators into stages of tasks. A stage is comprised of tasks based on partitions of the input data. Storing a job in a DAG allows for lazy computation of RDDs and allows Spark's optimisation engine to schedule the operations in ways that can significantly improve performance. The DAG is not evaluated until the user runs collect or another action, at which point the DAG is submitted to a DAG Scheduler. The DAG scheduler pipelines operators together e.g. many map operators can be scheduled in a single stage. The final result of a DAG scheduler is a set of stages. These stages are then passed on to the Task Scheduler which allocates workers via a cluster manager such as Yarn. The scheduler can allocate workers based on the data location to help improve performance. Workers

then preform the tasks on the Slave. In this way, each worker is only aware of the code that is was assigned. The DAG below was generated from our Fock matrix update.

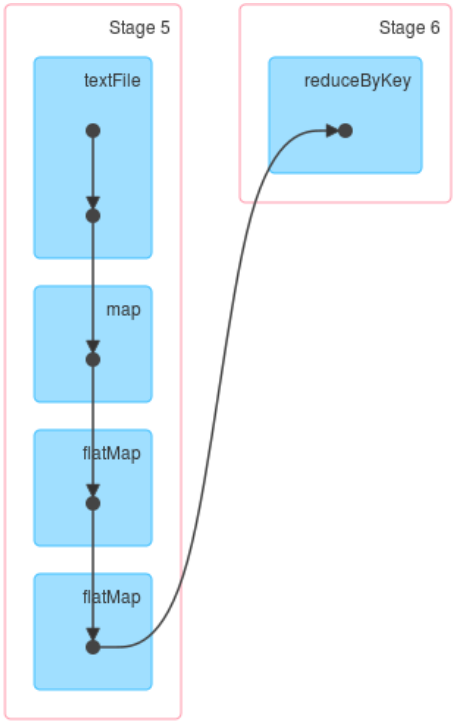



**Figure 2:** Directed Acyclic Graph for reading the 4d integral & using this with $D_{\mu\nu}$ to build the $F_{\mu\nu}^{new}$ matrix

## Results and Future Improvements

The Spark Prototype both worked for small basis sets such as $H^2O's$ STO-3G minimal basis set. However issues still remain for configurations with over 100 basis functions with integral files over 80 Mb, as the energy calculations diverge for these. The remaining problems still need to be tacked. Our conversion of the 4d integral binary file → HDFS txt splits a file into multiple partitions which we suspect the code is not handling correctly. There is also an issue with the calculations which involve higher angular momentum functions. These are incorrect when compared with standard libraries and causes the energy calculations to diverge. There may be other obstacles but these seem to be the main culprits.

To obtain a fully scalable Hartree-Fock implementation we need to check partition structure to ensure the integral is still being stored in the correct format in HDFS and create a union of multiple RDDs before preforming the F update step. Another way of improving the code might be to read the binary files to Parquet files[5] instead of HDFS which store data as columns of doubles rather than strings. This would make the reads on the datasets easier and potentially faster as Spark is optimised for this format. The energy divergence suggests that the Libint style calculation of $F_{new}$ also needs to be reviewed and debugged. Once these issues are fixed we can start to fully exploit Sparks ability to deal with large distributed datasets and measure it's performance and fault tolerance in comparison to previous implementations in C and Fortran.

## Conclusion

Over the course of the project the Hartree-Fock method from the quantum chemistry libint C libray was implemented in Scala before being adapted to Spark. The core component of the program was performed in a functional rather than object-oriented manner. Though the prototype worked for small basis sets we have been unable to resolve the remaining issues in time to allow the code to run at full scale with Gb integral files. The areas that need to be adjusted have been identified, unfortunately due to earlier delays we did not have time to implement them. Despite the set backs we still think that Spark is a useful tool for large-scale data processing.

## References


[1] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. (2012).

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12),

[2] Libint2, E. F. Valeev "A library for the evaluation of molecular integrals of many-body operators over Gaussian functions", http://libint.valeyev.net/

[3] Apache Spark "A general engine for large-scale data processing." https://spark.apache.org/

[4] Scala: cross-paradigm programming language, supporting both function & object-oriented methods http://www.scala-lang.org/

[5] Apache Parquet "A Columnar storage format available to any project in the Hadoop ecosystem" https://parquet.apache.org/


PRACE SoHPCProject Title

Apache Spark: Bridge between HPC and Big Data?

PRACE SoHPCSite

Slovak Academy of Sciences, Slovakia


PRACE SoHPCAuthors

Oisín Benson, Ireland

PRACE SoHPCMentor

Doc. Mgr. Michal Pitonak, PhD., Computing Centre of the Slovak Academy of Sciences


Oisín Benson


PRACE SoHPCContact

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


PRACE SoHPCSoftware applied

Spark, Scala, Breeze

PRACE SoHPCMore Information

www.spark.apache.org


PRACE SoHPCAcknowledgement

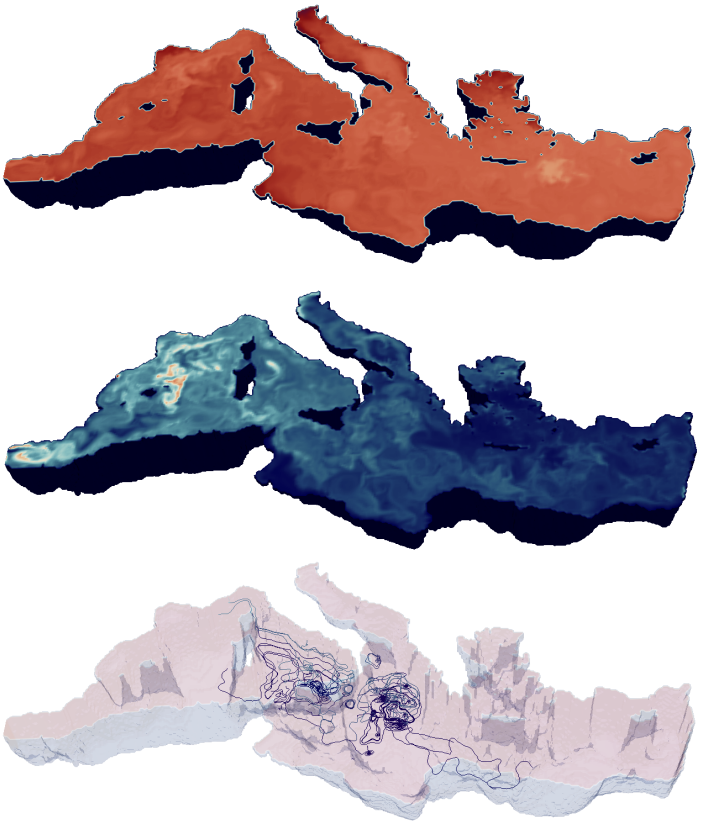
I'd like to thank my supervisor Michal Pitoňák for all his help and advice throughout the project. I'd also like to thank master sys admin Lukáš Demovič for helping to set up the various tools I needed for the project.

PRACE SoHPCProject ID

1606

# Real Time Exploration of Data

*Petr Valenta*



In situ visualisation technique allows scientists to explore data while a simulation is running. Furthermore, they can adjust its parameters and observe the immediate impact on the studied phenomena. This accelerates computation and provides much better insight than traditional approaches.

Traditionally, the process of performing numerical simulations consists of three separate steps. First, the input parameters (such as initial or boundary conditions) are specified, then the simulation is executed and finally the generated data are explored to determine the result. Throughout the years, advances in parallel computing methods coupled with the increasing power of supercomputers have allowed scientists to perform more accurate simulations in various fields of human research.

This increasing demand for simulations though requires for more data to be transferred and stored. The transfer of all the generated data for later analysis though requires too much time, so in practice data is stored only at several time-steps or using a lower resolution than the original data. This "discarding" of data could potentially mean that information is lost.

## What is in situ visualisation?

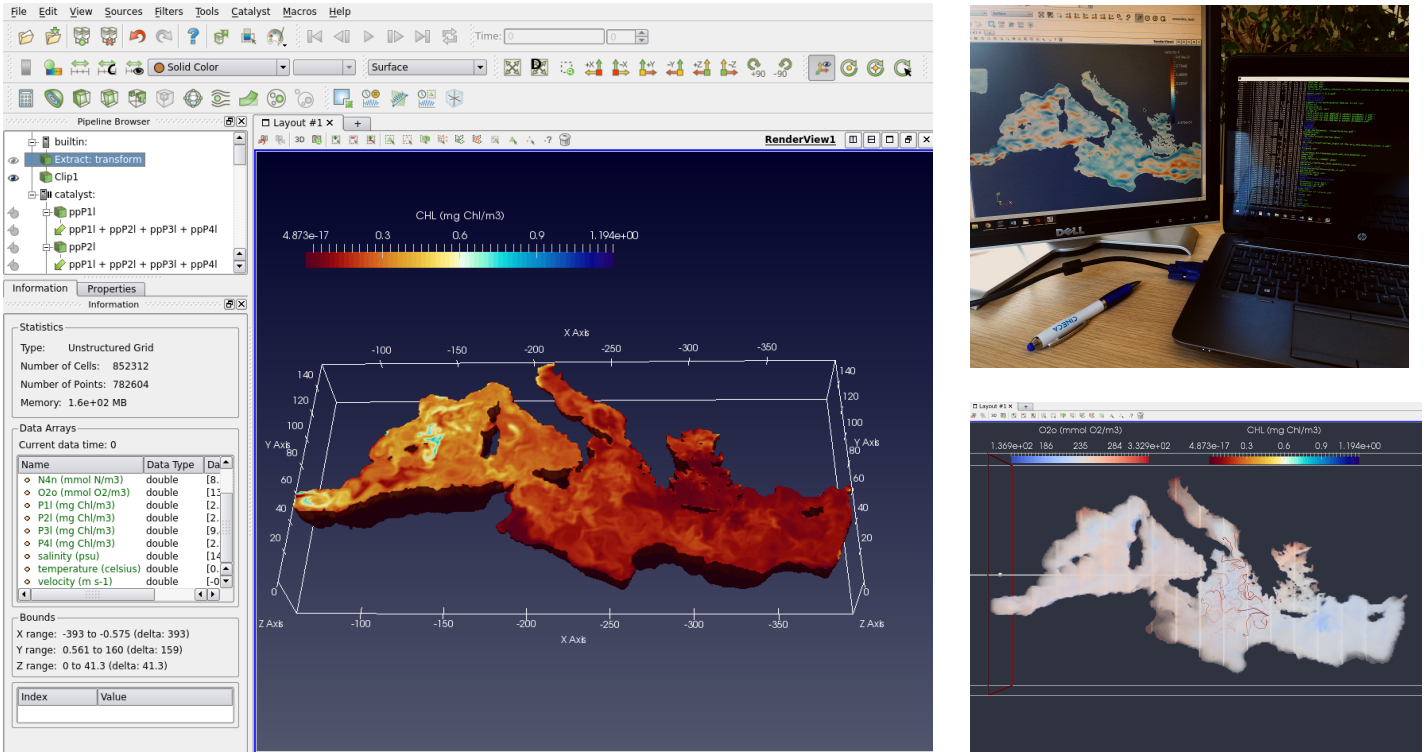
In situ visualisation describes techniques where data can be visualised in real-time as it is generated during a simulation and without it being stored on a storage resource. By coupling the visualisation and simulation, the data transfer bottleneck can be overcome. Furthermore, this approach allows scientists to monitor and interact with a running simulation, allowing for its parameters to be modified and allowing to immediately view the effects of these changes.

## How to instrument my code?

Significant development of several in situ solutions already exist and this can be embedded in simulation code. This project considered ParaView,[1] a popular multi-platform scientific data analysis and visualisation environment which is distributed under an open source license. ParaView was mainly used for post-processing of extremely large datasets, but now - thanks to the Catalyst library, ParaView can also be used as an in situ visualisation tool.

## Adaptors and pipelines

Catalyst, a relatively new component of ParaView, has been designed for fast integration with numerical codes to allow for real-time analysis of generated data by specifying which data needs to be visualised and analysed in situ. For this reason, Catalyst uses pipelines that are executed during the initial phase of the numerical simulation which allows for the post-processing capabilities of ParaView to be utilised. In other words, the data which the simulation will produce




Example of simulation connected to Catalyst (image on the left). In the top-left window you can see the Catalyst sources and the datasets that are extracted to the server. Real-time results are then visualised in the main window. A user can observe the data as it is being generated using visualisation nodes of supercomputer (top-right image).

are selected, then filters are applied and finally the data that should be dumped for deeper investigation are chosen. In this way, the output data can be significantly reduced because the processed elements, which carry all the interesting information are much smaller than the full datasets (Fig. 1).

Since ParaView is built on the standard visualisation toolkit VTK, the simulation internal data structures have to be transferred into the VTK data structures. This is achieved using simulation interfaces called adaptors - which should be separated from the code to simplify the build process. Three functions of the adaptor need to be called from code. The first one is called only once per simulation run and it initialises Catalyst and loads preconfigured pipelines. The second one creates VTK grids, appends the computed attributes on it and dumps selected elements with the frequency specified by the user. The last function is called at the end of the simulation to release all Catalyst resources.

## Time to run a simulation

Once the above function calls are specified and the code is coupled with Catalyst, it is time to run the simulation. pvserver - a component representing the server on which ParaView is running to process all the data, should first be initiated. Using a ParaView client, one can connect to this server to control the visualisation remotely from a local machine or using visualisation nodes of a supercomputer. The main advantage of using this is that pvserver can be executed in parallel, so with enough resources even extremely large datasets can be explored smoothly. Using ParaView client a connection to Catalyst can be created and then pvserver waits for the simulation data. The last step is to execute the simulation. Notice that the order is not important, there is no problem of connecting to already running jobs.

While a simulation is running, a user can see the size of the datasets that a simulation produces. But none of this data is physically stored on a storage system. The computationally expensive operations are carried out using ParaView's graphical interface. So, the user can select data structures and analyse them in the same way as in post-processing - which requires the saving of datasets onto a file system. But there is one difference, the simulation is in progress so a user can observe the data as it is being generated. With Catalyst, it is also possible to pause the simulation or specify a break-point at a selected time step. This can be helpful if a user expects some interesting behaviour of investigated phenomena or for identifying regions where numerical instability arises.

## Real world example

Researchers at OGS (National Institute of Oceanography and Experimental Geophysics) use a 3D numerical state-of-the-art model called OGSTM–BFM to study the nutrient and carbon cycles and their sensitivity to climatic changes in the Mediterranean.[4,5] The model computes biogeochemical fluxes which transform organic and inorganic components and can be used to predict the environmental health state of the Mediterranean. The OGSTM-BFM model is currently used to produce forecasts of the biogeochemical state of the Mediterranean for the European Copernicus Marine Environment Monitoring Service[3] (CMEMS). A recent requirement to increase the spatial and temporal scales of the simulations encountered aforementioned problems leading to longer computation times and difficulty to analyse produced data.

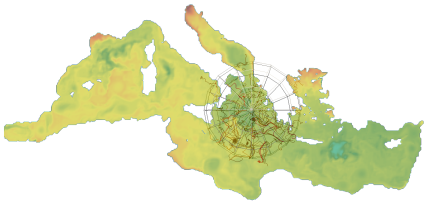


**Figure 1:** A user can dump elements such as slices or streamlines in order to speed-up simulation and save storage resources.

During the two months of my project stay at CINECA supercomputing centre, the OGSTM-BFM model has been coupled with ParaView Catalyst to enable real time analysis. Scientists now have a 3D in situ visualisation tool available to them, which can be used to check and analyse the OGSTM–BFM model behaviour by consistently evaluating how the biogeochemical processes are influenced by the nutrient and carbon cycles, specifically related to the three main boundary conditions of the Atlantic inflow at Gibraltar Strait, the terrestrial inputs at rivers and the atmospheric deposition. The tool is also beneficial to control the correctness of the computations during the simulation runtime. Furthermore, the implementation has been designed in such a way that allows portability to other coupled modelling systems used at OGS for many different purposes.

## Are there any drawbacks?

During the work, several performance tests were performed. It turned out, that there is a constant memory overhead (around 2.5 GB) caused by linking ParaView Catalyst libraries. Nevertheless, the required amount of memory is easily manageable using available supercomputers. However, each data array of the quantities chosen to be explored in situ have to be converted into the VTK field and the memory for these fields has to be allocated even if they are not actually observed or dumped. In this case, the additional memory resources may be significantly higher and the simulation code developer should be aware of this.

For instance, the OGSTM-BFM model computes up to 50 biogeochemical concentrations. The difference between the initial and instrumented code which exports all of the concentrations was in the case of simulation with a domain size $400 \times 400 \times 43$ more than 25 GB (growth from 20 GB to 45 GB). For comparison, the current domain size of the Mediterranean simulations in production phase is $1085 \times 480 \times 130$ (almost 10 times more grid points) and is expected to be further increased in the near future.

Significant amount of the memory could be saved, if each concentration is assigned to a single grid. This approach however was not convenient in our case because if the user decides to observe the data structure in a ParaView client, all the data has to be transferred. This operation could be computationally expensive and very long. Moreover, if the ParaView client is not executed in parallel, the memory of the visualisation node might not be sufficient and the ParaView client will likely crash. The elegant solution could be a dynamical grouping of quantities onto several grids specified by the user's demand via ParaView GUI. This issue is currently considered and will be figured out soon.

The next question which should be taken into account is whether it is worth performing a live visualisation. Sometimes even a single computational time step may be very long, thus the scientist cannot register any significant change. In this case, it is better to configure pipelines to render images and a user can create an animation afterwards during post-processing - however the nice feature of real-time insight and steering of the simulation is not available.

With regards to the computational expense of an in situ visualisation, there is no significant drawback. Simulation can be slowed down during the initialisation phase, when all the grids have to be created but beyond that there is not much difference. Several cycles are obviously spent on a visualisation, however these would have been spent for the output operations anyway.

## Conclusion

In summary, ParaView Catalyst is easy to integrate with already existing simulation code and offers a really deep insight into the large amount of data corresponding to simulated phenomena. If code spends too much time dumping generated data or if there are insufficient storage resources, ParaView Catalyst could be an elegant solution.

In situ visualisation creates an opportunity to explore and analyse much more data than is possible with traditional techniques and is expected to enable a wide range of new interactive applications in the future. In addition, as high performance computing moves towards the exascale era, the in situ approach is widely predicted to become more and more important as an efficient tool for speed-up of large scale simulations.[2]

## References


[1] Utkarsh, A. (2015) The ParaView Guide: A Parallel visualisation Application *Kitware*

[2] Bethel, E. W., Childs, H., Hansen, C. (2012) High Performance visualisation: Enabling Extreme-Scale Scientific Insight *Chapman & Hall/CRC*

[3] Copernicus Marine Service http://marine.copernicus.eu/services-portfolio/access-to-products/?option=com_csw&view=details&product_id=MEDSEA_ANALYSIS_FORECAST_BIO_006_006

[4] Lazzari, P., Teruzzi, A., Salon, S., Campagna, S., Calonaci, C., Colella, S., Tonani, M., Crise, A. (2010) Pre-operational short-term forecasts for the Mediterranean Sea biogeochemistry *Ocean Sciences*, 6, 25-39. doi:10.5194/os-6-25-2010

[5] Lazzari, P., Mattia, G., Solidoro, C., Salon, S., Crise, A., Zavatarelli, M., Oddo, P., Vichi, M. (2014) The impacts of climate change and environmental management policies on the trophic regimes in the Mediterranean Sea: Scenario analyses *Journal of Marine Systems*, 135: 137-149.


PRACE SoHPCProject Title

In Situ or Batch visualisation of Biogeochemical State of the Mediterranean Sea

PRACE SoHPCSite

CINECA, Italy

Petr Valenta


PRACE SoHPCAuthors

Petr Valenta, Czech Republic

PRACE SoHPCMentor

Paolo Lazzari, OGS, Italy
Stefano Salon, OGS, Italy

PRACE SoHPCContact

Petr Valenta
Phone: +420 606 489 018
E-mail: valenpe7@email.cz


PRACE SoHPCSoftware applied

OGSTM-BFM, ParaView, Blender

PRACE SoHPCMore Information

summerofhpc.prace-ri.eu


PRACE SoHPCAcknowledgement

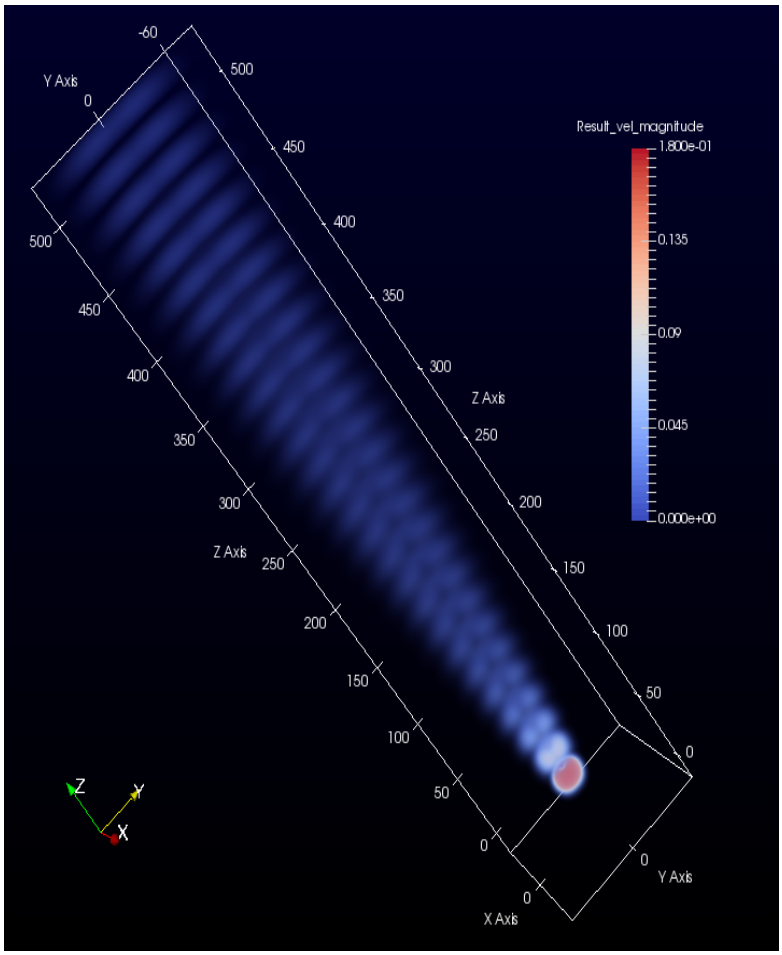
I wish express my gratitude to my supervisors Paolo Lazzari and Stefano Salon, to the site-coordinators Luigi Calori and Massimiliano Guarrasi, and also to Eric Pascolo, Silvano Imboden and Daniele de Luca for constant support and guidance, as well as for providing invaluable advice and direction during my stay at CINECA.

PRACE SoHPCProject ID

1607

# InSitu visualisation Tornado effect

*Anurag Dogra*




InSitu refers to real time visualisations and in this project we looked into the real time visualisation of the velocity fields and enstrophy of a tornado using the given blow up NS code and its generated data. ParaView was used for live visualisation.

This project is a numerical study for the solution of the Navier-Stokes equations upon smooth initial data. These mathematical models have a wide range of applications and in this project we looked at tornadoes. A tornado is usually a rotating column of air which is in contact with the ground and a cloud base. We simulated the evolution of a 3 dimensional vector field in real space $\mathbb{R}^3$ and found the suitable input data for the simulation which produced the velocity fields and enstrophy values which could be visualised using ParaView Catalyst. The best part about using Catalyst is its live visualisation capability, thus there is not need to store data for post processing - thus saving on storage space.

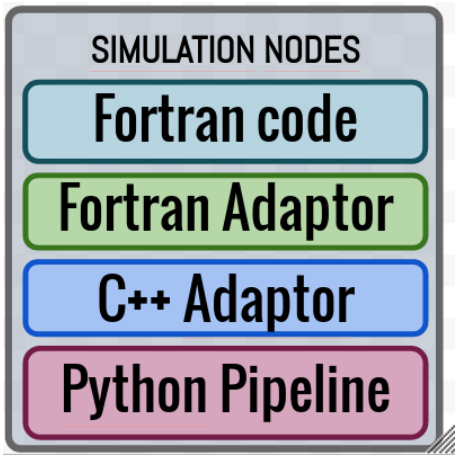
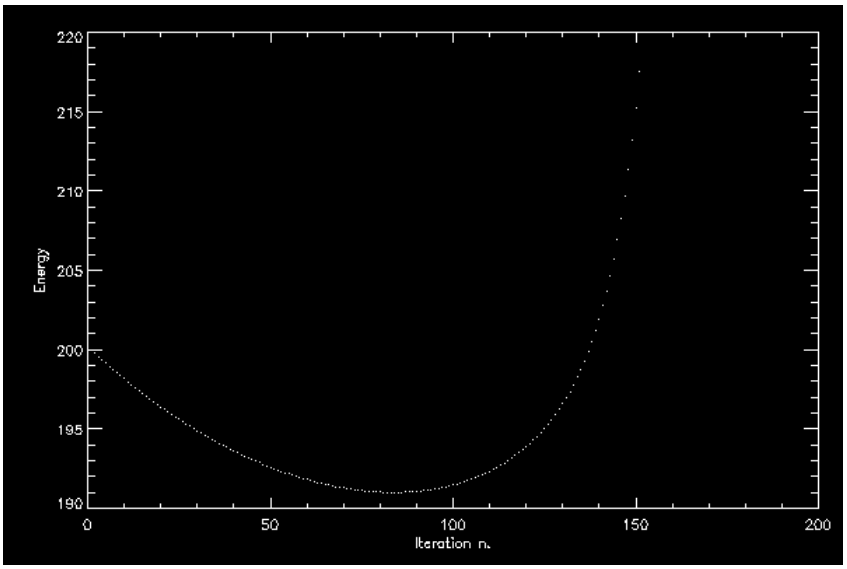



**Figure 1:** Program structure for ParaView Catalyst

## Catalyst Program structure

To work with Catalyst we had to write adaptor functions for the simulation code. The overall structure of the code is the main Fortran simulation code with a FORTRAN adaptor code which diverts velocity parameters to a C++ adaptor function (responsible for creating the grid like box structures) while simultaneously checking whether ParaView catalyst data requests are required by the visualisation node.The Python pipeline works as communicator which selects data from a simulation and contains the address and port of the visualisation client where ParaView is running.



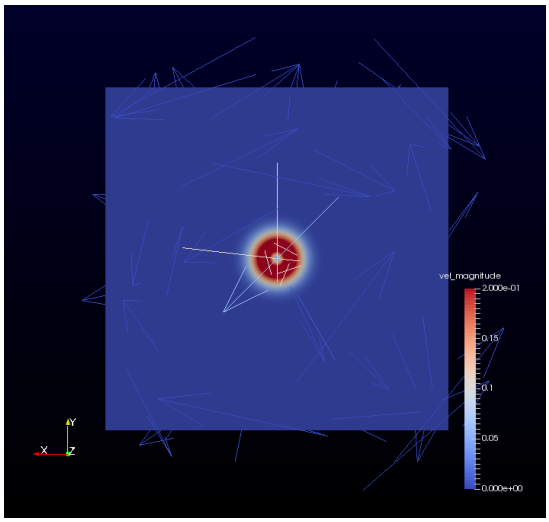




**Figure 2:** Energy vs time(iterations) and Velocity magnitude fields at 150 iteration

## Assumptions for simulations

Our simulation is related to the complex solutions of the Navier-Stokes equation:[2]

$$\frac{\partial \mathbf{u}}{\partial t} + \sum_{j=1}^{3} u_j \frac{\partial}{\partial x_j}\mathbf{u} = \Delta \mathbf{u} - \nabla p \quad (1)$$

where

- **u** is the velocity field
- p is the pressure
- *t* is the time

A solution for a finite time blow up problem was proven in[3] and can thus be reformulated into a convolution integral equation by introducing the modified Fourier transform **v** and **u**.

For the simulation we consider the non-zero infinite system and the finite region A as our domain.This means that we assume:

$$\mathbf{v}(\mathbf{i}) = 0 \qquad \mathbf{i} = (x, y, z) \quad (2)$$

where **i** does not belong to domain A, which implies that our simulation is comparable to real phenomena if during simulation time the value of the true vector field outisde domain A is approximately equal to 0.
Also considering the velocity increment along the z direction our main simulation code was formulated on the basis of these energy and enstrophy equations:[1]

$$E(z,t) = \frac{1}{2}\int_{\mathrm{R}x\mathrm{R}} |\mathbf{v}(\mathbf{z},t)|^2 dxdy \quad (3)$$

$$S(z,t) = \int_{\mathrm{R}x\mathrm{R}} |\mathbf{z}|^2 |\mathbf{v}(\mathbf{z},t)|^2 dxdy \quad (4)$$

- E(z,t) is energy along z axis
- S(z,t) is enstrophy along z axis

with similar equations applying for the x and y axis. Figure 2 depicts a graph of the enstrophy vs time with domain size 121 X 121 X 548. An increase in the energy values between 100 to 150 depicting an increase in the velocity fields at that time range is also shown.

## Domain Distribution

The x,y,z box domain has to be divided into the number of available cores using a 2D x-pencil decomposition FFT library. For our simulation, y-axis and z-axis points were divided across 8 cores and x-axis were available on all cores as shown in **Table 1**.

## Discussion on Results

After the perfect domain size, simulations were run on the iteration range 100-150 where an increase in energy was noted. Connecting to the catalyst at iteration 105 we noticed the initial velocity field resembling a donut as shown in **Figure 4** because at this iteration energy increases very slowly. The best feature of ParaView catalyst is that one can connect to it at any time while simulation is running and there is no need for the simulation to finish to see a visualisation.

**Table 1:** Domain Distribution

| Axis | Domain Size | Divide in cores |
|---|---|---|
| x | 121 | 1 |
| y | 121 | 8 |
| z | 548 | 8 |

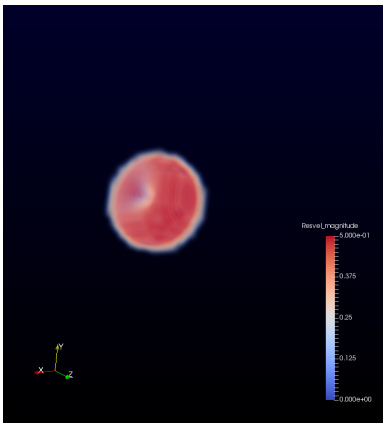



**Figure 3:** Magnitude of velocity field

Connecting to catalyst between 140-150 iterations, the final observed enstrophy looks like increasing donuts along the z axis, which means there is an increase in velocity fields as well as in enstrophy along the z axis towards the other end of the box.

We have defined our Gaussian center at (0,0,20), 20 along the z axis which is why we can see donuts generated after every 20 steps. This means that at a particular place we have velocity equal to zero as shown in the **Figure 3**.

When these donuts reach the boundary of the domain along the z axis, values of enstrophy at the boundary is dif-

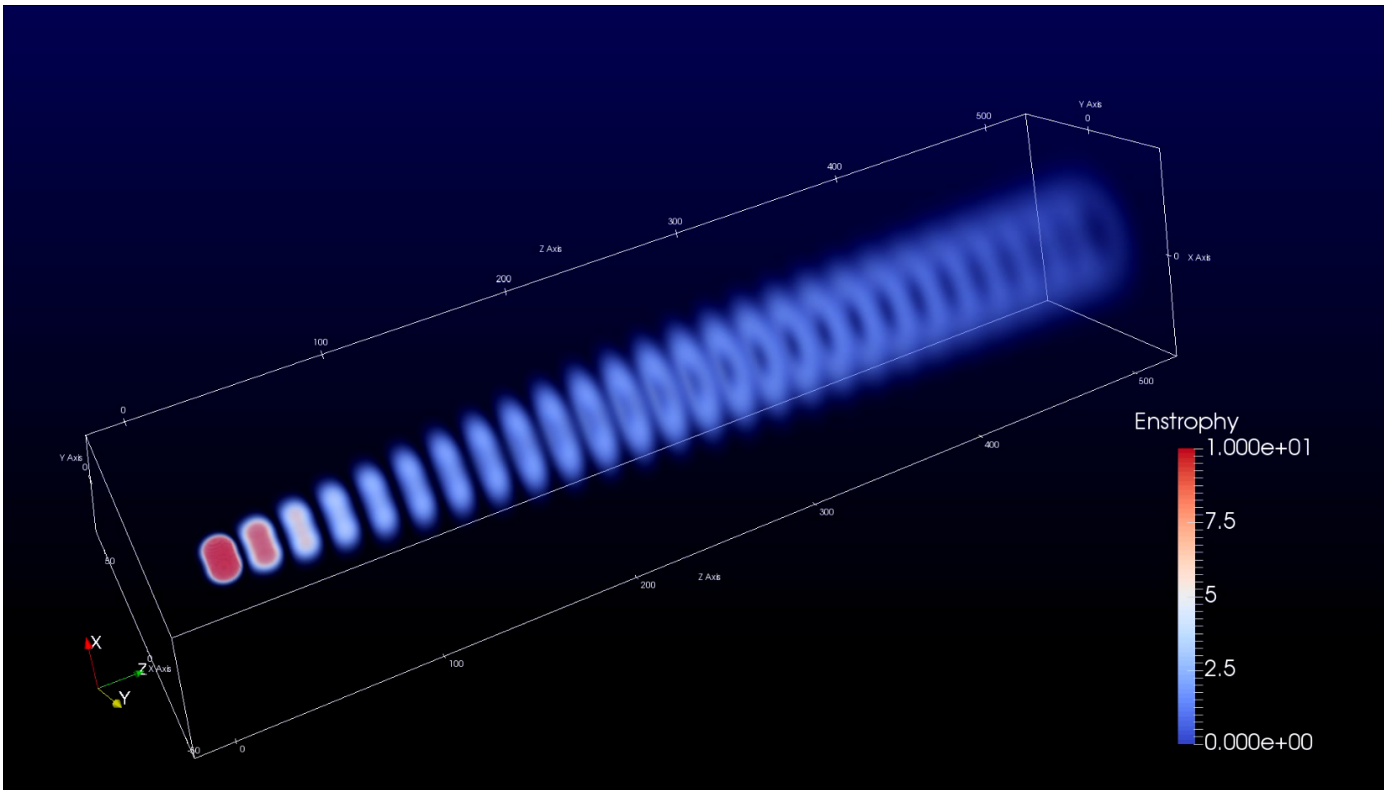



**Figure 4:** Enstrophy at 150th iteration

ferent from zero and presumably also outside the domain.This means our assumption at this time step is not correct any more which implies that we have to look only before that time step.

## Conclusions

We have concluded from the visualisations of this simulation that for every finite region we can only reach near the explosion time.

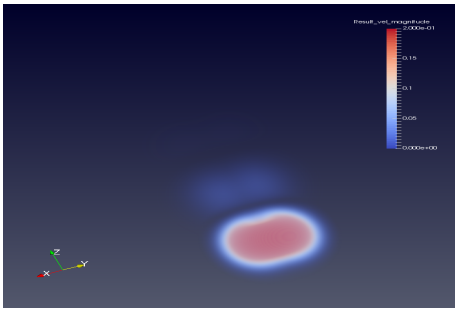


**Figure 5:** Increasing Velocity magnitude in between the simulation

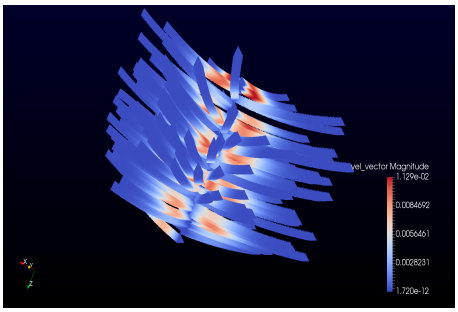


**Figure 6:** Stream tracer of velocity magnitude in near the initial part of domain along z axis

## References

[1] IMA Journal of Applied Mathematics(2005). Complex singular solutions of 3-d Navier-Stokes Equations: and related real solutions. doi: 10.1093/imamat/hxh000

[2] LERAY,J.(1934) Sur le mouvement d'un liquide visquex emplissant l'éspace. *Acta Math*,63,193-248

[3] LI,D,'|&'SINAI,YA.G.(2008) Blowups of complex solutions of 3D Navier-Stokes system and renormalization group method. *J. Eur. Math. Soc.*10,267-313.

### PRACE SoHPCProject Title

InSitu visualisation of Navier- Stokes Tornado Effect

### PRACE SoHPCSite

CINECA supercomputing center, Italy

Anurag Dogra

### PRACE SoHPCAuthors

Anurag Dogra, [TU Bergakademie Freiberg] Germany

### PRACE SoHPCMentor

Sandro Frigio,Università di Camerino, Italy

### PRACE SoHPCContact

Anurag, Dogra, CINECA
Phone: +4917680823911
E-mail: anuragdogra.2192@gmail.com

### PRACE SoHPCSoftware applied

ParaView 5.1,Blender-2.77a,GIMP Image Editor version 2.8.10-0ubuntu1.1 and Audacity Version 2.0.5-1ubuntu3.2

### PRACE SoHPCMore Information

www.paraview.org www.blender.org www.gimp.org www.audacityteam.org

### PRACE SoHPCAcknowledgement

Thanks to Sandro Frigio,Luigi Calori,Massimiliano Guarrasi,Daniele De Luca and Silvano Imboden for their support.

### PRACE SoHPCProject ID

1608

Bored with slow applications and eager for the results? Check this out!

# Python scientific application booster

*Marta Cudova*


Python is a modern programming language bridging the gap between scientists and IT specialists. Python enables rapid prototyping of realistic simulations and brings the power of supercomputers closer to everyday science. Research impossible in the past now turns into reality.

Computer simulations have become an essential part of modern science. Most of the current research would not be possible without running extensive simulations for various reasons, such as financial expenses (car crash tests), environment impact (nuclear tests), ethical issues (medical research), or simply its practicability (collisions of galaxies).

However, to satisfy the computational requirements of such simulations, our ordinary computers are not fast enough. We need a supercomputer. Such machines harnesses the power of thousands of tightly connected high-end servers to deliver enough power to our software. One of such machines is Archer located in Edinburgh. Archer is the UK's leading supercomputer ranking among fifty of the fastest supercomputers in the world this year.

Nevertheless, creating a realistic simulations is not an easy job. The simulations have to be partitioned into smaller tasks, implemented in low-level languages, optimized for the target machine, thoroughly tested, experimentally validated and executed in parallel using special libraries called `MPI`. This process is usually very time consuming and requires deep experience in high performance computing (HPC).

Naturally, not every scientist can also be an excellent HPC developer. We need a solution to overcome this gap - a modern high level language which would make the development much easier. One such a language is Python. Python is considered to be a suitable connection between scientists and HPC programmers. It excels in rapid prototyping with a very fast learning curve. Python also offers a lot of specialised packages for scientific computing (e.g., `numpy`, `scipy`), high performance computing (e.g. `mpi4py`) and visualisation (e.g. `matplotlib`).

In this report, I would like to show you how to solve computationally intensive problems in Python and get you in touch with HPC. I selected two toy applications, image reconstruction and computation fluid dynamics. In both cases, I will start from a serial version running on a single processor core. This version will be implemented in Python and some other low-level language. Next, I will accelerate the application by distributing the work over a thousand processor cores communicating between them using the message passing interface (MPI) library. Finally, I will compare all developed codes. You will be impressed by how fast my applications can run, but also understand their limits.

## Study 1: Image Reconstruction

### Introduction

In this case study, we take an image processed by an edge detection algorithm and try to reconstruct the original one. The reconstruction is done iteratively by gradually improving the image until its quality reaches a desired level, see Fig. 2. At the beginning, the image is divided into tiles and distributed over processor cores. All cores calculate their tiles and exchange necessary information to keep the image consistent. After completion, the tiles are gathered to create the reconstructed image. The communication is orchestrated by the Message Passing Interface (MPI).

**Figure 2:** The image reconstruction visualisation. (a) Input image processed by edge detection and (b) reconstructed image.

### Methods

Since the image is represented by a large two-dimensional array of pixels, we can use geometric decomposition to divide the work over multiple processor cores. Fig. 3 illustrates 1D (stripe) and 2D (tile) decompositions.

The reconstruction algorithm is quite simple. It iteratively processes the image, calculating the new colour of each pixel from its neighbourhood and the original pixel, see Eq. (1).

$$new_{i,j} = \frac{1}{4}(old_{i-1,j} + old_{i+1,j} + old_{i,j-1} + old_{i,j+1} - orig_{i,j}) \quad (1)$$

where `old` and `new` are pixel colours at iteration $t$ and $t-1$ while `orig` is the colour of the processed pixel in the original image.

To reconstruct the image properly, the processors have to collaborate and exchange pixels at the tile edges with their neighbours. The tiles are thus made slightly bigger to accommodate pixels coming from the neighbours. These pixels are referred to as halo zones or overlaps. If the communication fails, the image shows various artifacts (see blurred stripes in Figure 3).

**Figure 3:** Different image decompositions.

### Implementation

I implemented the image reconstruction in C and Python, parallelised it using MPI and investigated several communication strategies and image decompositions. I focused on blocking and non-blocking communications, virtual topologies and rank reordering. So far, I have completed the implementation, validation, and performance evaluation of the 1D decomposition. The 2D decomposition is currently being tested. The performance was evaluated on an image of 768 × 768 pixels, see Figure 1. I measured the performance using various number of processor cores ranging between 1 and 768.

Furthermore, I extended the Fortran and Python reconstruction algorithm to support colours. These versions were used to generate animations illustrating the progress of reconstruction.

### Results & Discussion

Figure 1 shows the dependency of the execution time, speed-up factors and parallel efficiency on the number of processor cores used. For image reconstruction, linear scaling is expected. Thus when the number of processor cores is doubled, the execution time should be halved. We can see that this is the case for up to 64 cores. Beyond this point, the performance does not improve so much, the parallel efficiency drops and the speed-up reaches its plateau.

The first limitation is the communication overhead. This can be alleviated by the use of non-blocking communication which enables multiple network transfers to be executed simultaneously. My experiments showed that non-blocking communication can decrease the execution time by 50%. The second limitation comes into play as soon as the tiles become too small and the processors become underutilised. Let me note, when the image reconstruction runs on 768 cores, a single core only processes 768 pixels!

The experimental results also reveal Python and C codes were accelerated by a factor of 64 and 20 respectively. We can also see C codes being always faster, usually about 3 times. This is given by the C language, its compiler and better optimization for the underlying hardware. On the other hand, the Python code is much easier to read and extend.

The parallel efficiency shows the amount of overhead hidden in the execution time. It is general good practice to stop adding more processor cores once the efficiency drops below 50%. This happens when running the Python code on 128 cores and the C code on 32

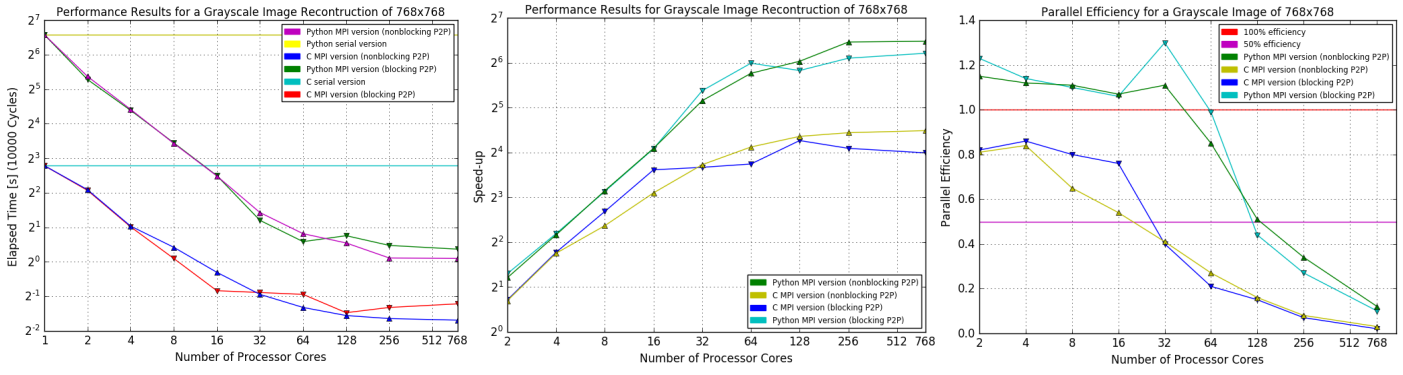



**Figure 1:** Performance comparison of C and Python codes: (a) execution time, (b) speed-up w.r.t. serial versions, (c) parallel efficiency.

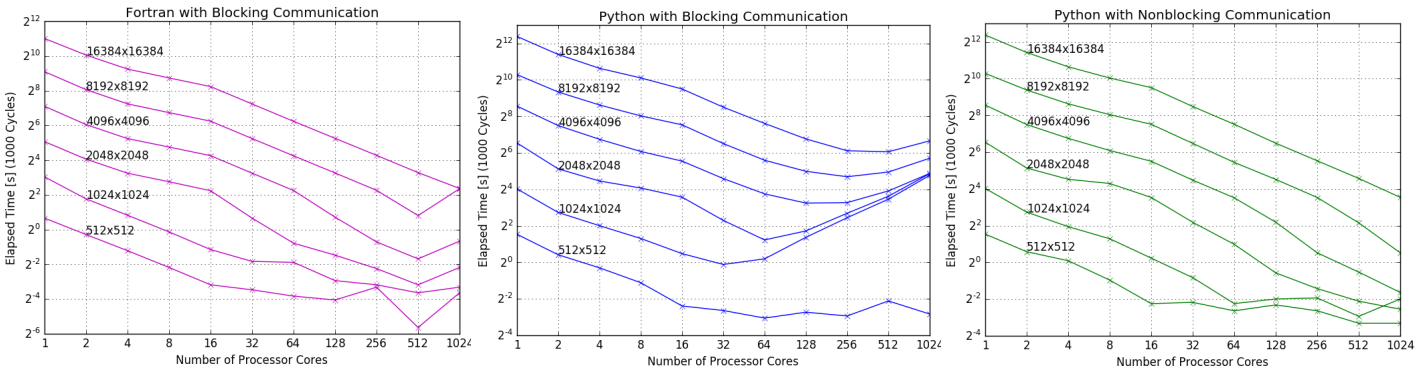



**Figure 4:** Execution times for various simulation domains for (a) Fortran CFD with blocking communication, (b) Python CFD with blocking communication and (c) Python CFD with non-blocking communication.

cores.

## Study 2: Fluid Dynamics

### Introduction

Computation Fluid Dynamics (CFD) studies the mechanics of fluid flow (i.e., liquids and gases in motion). It has a wide range of applications such as modelling F1 car aerodynamics, blood flow in coronal vessels, combustion processes in a Jumbo Jet engine. As expected, solving realistic problems requires supercomputers.

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

### Methods

The fluid flow can be described by the *stream function* $\psi$ defined as follows:

$$\nabla^2\psi = \frac{\delta^2\psi}{\delta x^2} + \frac{\delta^2\psi}{\delta y^2} = 0 \qquad (2)$$

When discretized using the finite difference approach, we can calculate the stream value at every grid point as:

$$new_{i,j} = \frac{1}{4}(\psi_{i-1,j} + \psi_{i+1,j} + \psi_{i,j-1} + \psi_{i,j+1} - 4 \cdot \psi_{i,j}) \qquad (3)$$

The computation iterates until the solution reaches the steady-state. This approach is called the *Jacobi Algorithm*.

Doesn't it remind you of something? Of course, this is a very similar approach to that used in image reconstruction.

In order to obtain the flow pattern of the fluid in the cavity, it is necessary to compute the velocity field $\tilde{u}$. The $x$ and $y$ components of $\tilde{u}$ are related to the stream function by:

$$u_x = \frac{\delta\psi}{\delta y} = \frac{1}{2}(\psi_{i,j+1} - \psi_{i,j-1})$$
$$u_y = -\frac{\delta\psi}{\delta x} = -\frac{1}{2}(\psi_{i+1,j} - \psi_{i+1,j}) \qquad (4)$$

This means the velocity of the fluid at each grid point can also be calculated from the surrounding grid points.

The parallel implementation lies in the domain decomposition. The grid is again divided into tiles and processed in parallel. The halo exchange is orchestrated by MPI.

### Implementation

I implemented the CFD problem in Fortran and Python, parallelised it using MPI and investigated several communication strategies and grid decompositions. I focused on blocking and non-blocking communications, virtual topologies and rank reordering. I succeeded with the implementation, validation, and performance evaluation of the 1D decomposition.

The performance was evaluated on various grid sizes between $512^2$ and $16384^2$ grid points and the number of processor cores between 1 and 1024. Furthermore, I created an animation demonstrating the process of the CFD calculation.

### Results and Discussion

Figure 4 demonstrates the strong scaling of parallel Fortran and Python codes using different MPI communication approaches. The strong scaling describes how the execution time is reduced when the number of cores is increased. It is evident, that the bigger the problem is the better the scaling (speed-up, efficiency) the codes reach, which is caused by more work assigned per processor core. The best scaling is observed for the Python non-blocking version. The curves look pretty narrow with a convenient slope.

On the other hand, the Python blocking code shows a very important phenomenon. Once the communication becomes the bottleneck, the execution time not only does not improve any more, but it actually deteriorates. Consequently, we must be careful when picking the optimal number of processor cores to run on.

## Final conclusion

Although Python is still falling behind C and Fortran in terms of performance, its main advantages manifest when prototyping new HPC codes. Thanks to Archer supercomputer, I could use 371 kAUs to develop and benchmark my codes. This corresponds to 1 year 4 months and 25 days on my laptop.

### PRACE SoHPCProject Title

Parallelising Scientific Python Applications

### PRACE SoHPCSite

Edinburgh Parallel Computing Centre, University of Edinburgh, UK

### PRACE SoHPCAuthors

Marta Cudova,
Brno University of Technology,
Czech Republic


Marta Cudova

### PRACE SoHPCMentor

Neelofer Banglawala, EPCC, UK

### PRACE SoHPCContact

Catherine, Inglis, EPCC
Phone: +44 131 651 3578
E-mail: c.inglis@epcc.ed.ac.uk

### PRACE SoHPCSoftware applied

Python, Anaconda, mpi4py

### PRACE SoHPCMore Information

www.python.org,
www.continuum.io/why-anaconda,
pypi.python.org/pypi/mpi4py,
www.archer.ac.uk,
mpi-forum.org

### PRACE SoHPCAcknowledgement

I would like to thank my project mentor, Dr. Neelofer Banglawala, for all advices and guidance she gave me. Then, I would also like to thank all EPCC staff, PRACE and coordinators from Juelich.

### PRACE SoHPCProject ID

1609

# Weather visualisation for outreach

*Tomislav Subic*

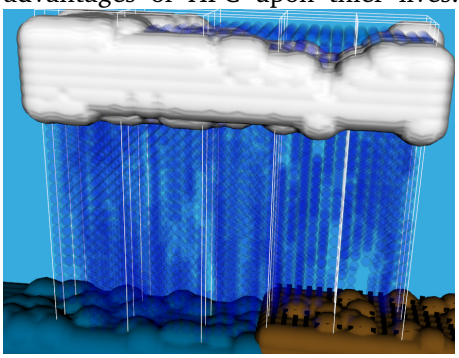

In order to bring HPC closer to the general public we will enable them to run their own weather forecasting simulations, something common both in their lives and in HPC.

In the scope of the project an application is developed which interacts with Wee Archie. It will be used at outreach events and in training courses. Users will be able to run their own weather simulations and see how different parameters affect the weather. How the cores work together and how this affects the performance will also be something that can be altered.

## Introduction

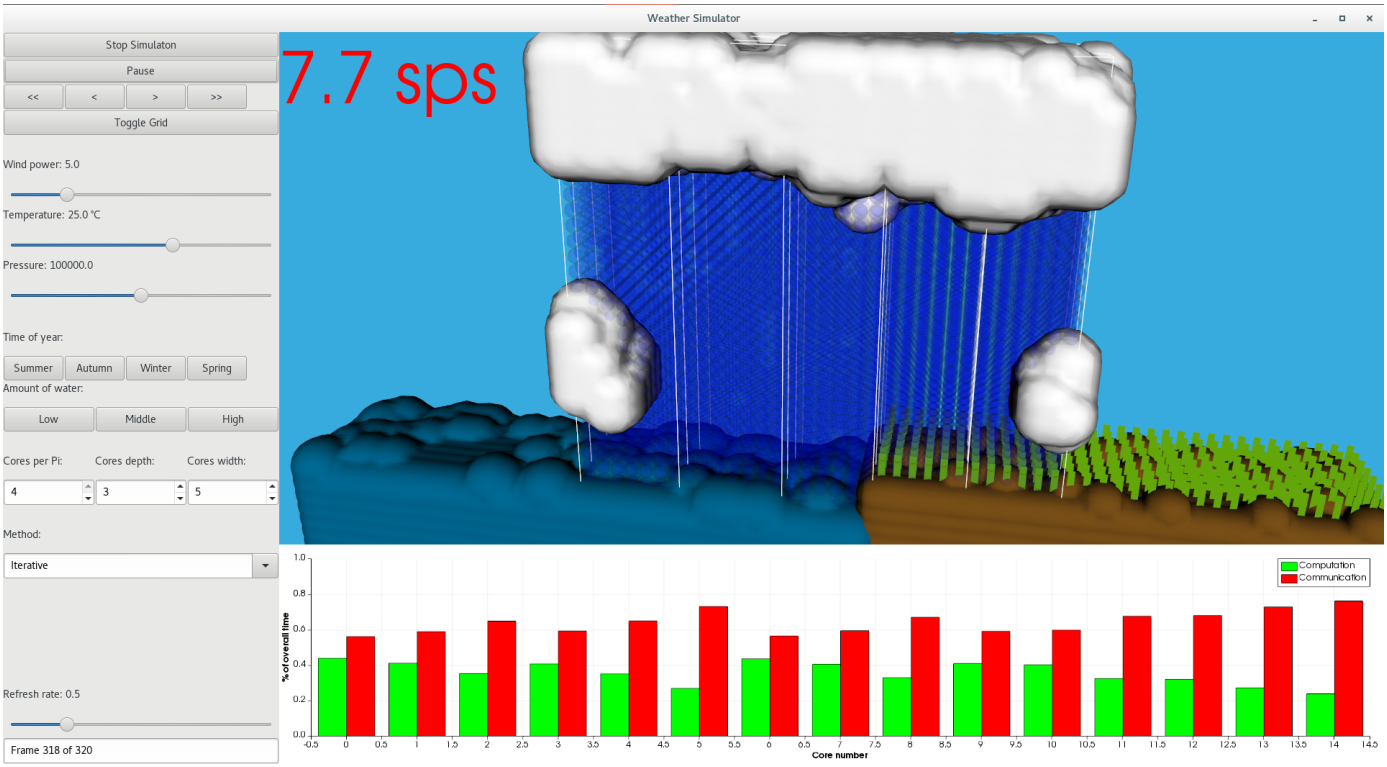
The main focus of this project is outreach - an important aspect for every field of science, so the general public follows the scientific community and the progress it makes. The benefits of outreach are two fold. With more outreach, people better understand the science behind weather forecasting and other research. People may also gain an interest in the field and decide to study it - thus growing the scientific community members. Another aspect of outreach is to share knowledge among the community and given that not everyone reads scientific papers, this can be achieved by organising outreach events and training courses.

High Performance Computing (HPC) is used by scientists from different fields who use computational models to describe real systems. As an example, it is more efficient for a chemist to run a few simulations before confirming the results with just one experiment. But simulations can require a lot of time on regular computers, so in order to speed them up, scientist use HPC and supercomputers. The terms HPC and supercomputers may seem abstract to a lot of people and in this project we aim to change that. The main motivation behind this project is to find a better way for HPC to engage with the general public and demonstrate the advantages of HPC upon thier lives.



A common and very important application of HPC is that of weather forecasting. EPCC and the UK Met Office have together developed a new state of the art weather forecasting model called MONC which the scientific community is beginning to use. MONC is a highly scalable Large Eddy Simulation (LES) model that has been developed to simulate clouds and turbulent flows. MONC can simulate the atmosphere at a very high resolution with an accuracy of up to 10s of meters which is something that no other model is capable of doing. Due to the way it is written, it is very scalable and suited for running on a very large number of cores. That is pretty impressive on its own, but can we show it to the general public? Can we use it to explain weather forecasting and the power of HPC? This project aimed to provide this answer. A weather visualisation application is developed to demonstrate weather forecasting and HPC. The user can specify simulation input parameters such as temperature, pressure and wind power, but also how many cores will be used and the way the workload will be divided among these cores. This resembles the way scientists run simulations, but also how computer scientists and HPC experts find different ways to speed up the computations.

In addition to these simulation parameters, there is also Wee Archie, a small supercomputer made out of Raspberry Pi's which reassembles ARCHER - the UK's main supercomputer. Wee Archie will be used at outreach events, to provide the general public a better understanding of supercomputers, how they are built and how they operate. Since MONC is so flexible, it has no problems running on Wee Archie and while simulations run, a visualisation will display the results in the form of a

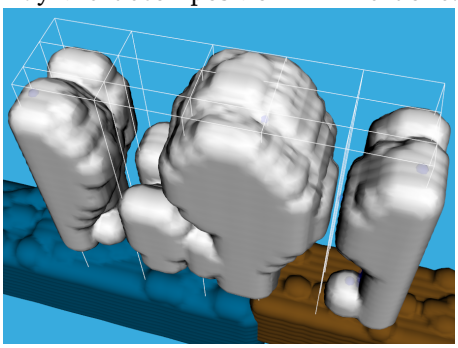



moisture field with different levels and concentrations defining clouds and rain. To display the load, each Raspberry Pi has a small LED panel displaying the load of each core and the network load.

## Development

As this project is targeted for outreach, it makes sense to use Wee Archie as the supercomputer for all the calculations. EPCC has been developing a framework for such demos which helps to execute jobs on Wee Archie which includes the transfer of configuration files, executing a job and getting the generated data back. This project is the first to use this framework and apart from developing the application, I helped to improve the framework and fix some bugs. A few tools are needed for Wee Archie demonstrations - a programming language, a GUI framework and a visualisation tool. For the programming language Python3 was chosen. The chosen GUI framework was WxWidgets - an open source, popular, lightweight and cross-platform interface. Finally, the Visualization Toolkit (VTK) - a robust and open source tool for scientific visualisation, was also chosen.

We now describe the general pipeline of the application. On the left hand side panel is where input parameters for the simulation can be chosen. Of course, meteorologists use more input parameters, but for demo purposes we include the 5 parameters of wind power, atmospheric pressure, temperature, time of year and water level. Other options which do not influence the outcome of the simulation, but the performance of it are also included and these may be of interest to computer scientists or beginners in HPC. These allow to select the number of cores per node the simulation will run on, but also the way the decomposition will be done.

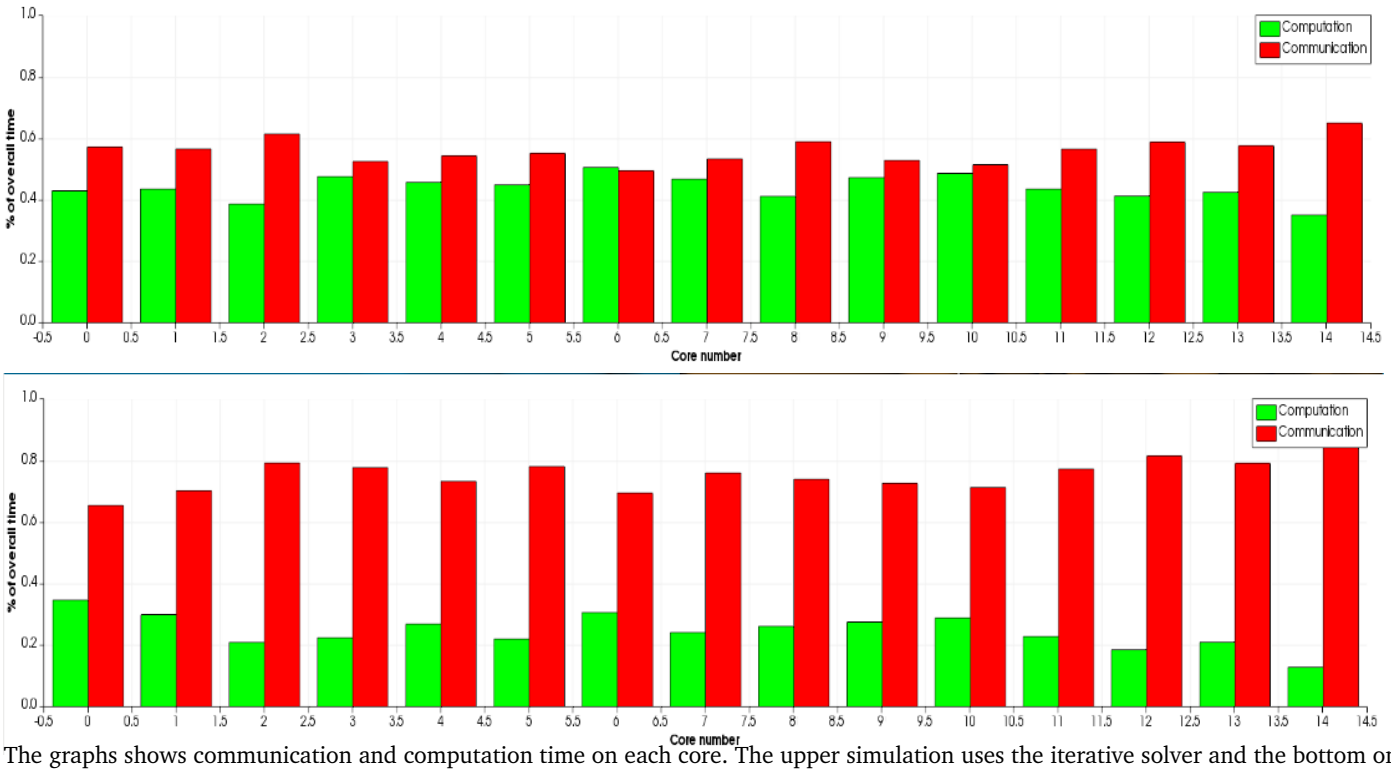


The simulation uses a 2-D decomposition technique to split up the workload amongst the nodes - despite the simulated space being 3-dimensional. Each core will then be assigned some piece of the atmosphere to work on, which will have a custom width and depth, but will take all of the height, ranging from the ground to the highest level of atmosphere. The number of processors upon which the workload will be spit can be set in both width and depth. To demonstrate performance, the user can also select one of two solvers for the Poison equations which determines pressure.

Once the selections are made, the configuration is sent to Wee Archie and it runs MONC. Every 5 steps (model seconds) a netCDF data file is generated and sent to the application. The file describes the moisture values for each part of the atmosphere (rain, clouds and vapour) and also provides the time each core has spent for calculation and communication.

A visualisation is generated each time Wee Archie generates a data file and VTK renders clouds using filters which create a surface around points in the atmosphere that have enough cloud mass. Rain is a separate quantity, the points in space which contain rain mass (moisture in rain state) are rendered as blue spheres. Transparency and colouring is applied from light blue to dark blue, depending on the amount on rain mass in a particular spot. Lots of rain is usually within the clouds, but we are unable to see that until it starts falling. Since clouds can form without any rain, clouds and no rain may be rendered. It is worth mentioning that the visualisation is meteorologically accurate as we can see how the cloud forms and moves. If we got the input parameters right, rain will accumulate around the clouds and start falling.

A gamification has also been included which simulates the effects of rainfall on crops. If enough rain falls, crops will rise out of the land, but too much rain can destroy the crops. A grid showing the decomposition is also ren-




The graphs shows communication and computation time on each core. The upper simulation uses the iterative solver and the bottom one uses FFT. We can see how FFT has less computation time and more communication time, but that does not mean that it is slower overall.

dered and this presents how the atmosphere is split up amongst the processes. The last part of the visualisation shows the computation time for each core as a green bar and the communication time of each core as a red bar. This allows to easily compare how much of the time is spent in communication and how much doing computation. Of course, it is of interest to tweak the settings so that we have low communication time and high computation time. Another performance measure appears in the upper left corner, which shows how many simulation seconds are computed in one real time second.

## Discussion and Conclusion

During the project I have developed an outreach application which can be used to describe simulations, HPC and parallelism. At the beginning of the project the idea was to develop a weather visualisation demo, but after a few discussions with my mentor we realised that this has a lot of potential. So apart from the clouds and rain, I decided to expand the project and display some other interesting information about performance and how parallelism is achieved.

Further to an outreach event application, it is perfectly suited to be used as an education tool, which some people are already planning to use as such. It can be used in regular classes or in training courses for both young meteorologists and other scientists who want to use HPC and computational methods to understand the impact in results and performance that these can impose. Most scientists who start to use HPC are not aware of the trade-offs they have to make and this application can give them an idea that performance in HPC depends on a number of various factors. The decomposition grid is a good way to show how the data and the workload is distributed among the cores. With different settings and performance results we can easily show that running simulations on a large number of cores is not enough and that we have to find the best way to distribute the workload.

Since this is the first demo application that runs on Wee Archie, it will also be used as a basis to develop other demo applications. Because of the way it is written and the tools it uses, it is also very easy to improve and change in the future - with different versions targeting different audiences. It will be interesting to see how this application will be used and what other outreach applications it can inspire.

PRACE SoHPCProject Title

Weather forecasting for outreach on Wee Archie supercomputer

PRACE SoHPCSite

EPCC, UK

PRACE SoHPCAuthors

Tomislav Subic, University of Rijeka, Croatia

PRACE SoHPCMentor

Dr. Nick Brown, EPCC, UK

Tomislav Subic

PRACE SoHPCContact

Dr. Nick Brown, EPCC, UK
E-mail: n.brown@epcc.ed.ac.uk

PRACE SoHPCSoftware applied

The Visualization Toolkit(VTK)

PRACE SoHPCMore Information

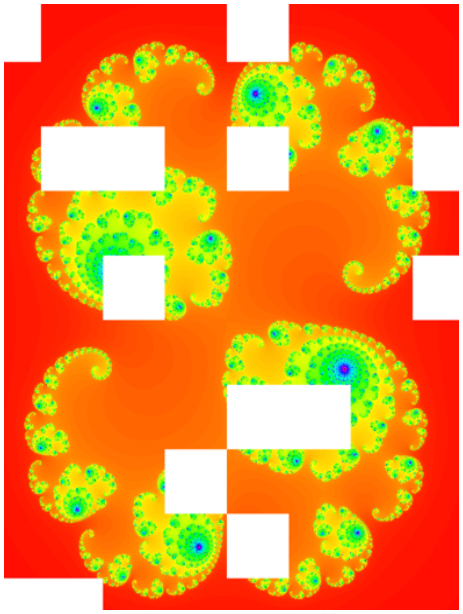
www.vtk.org

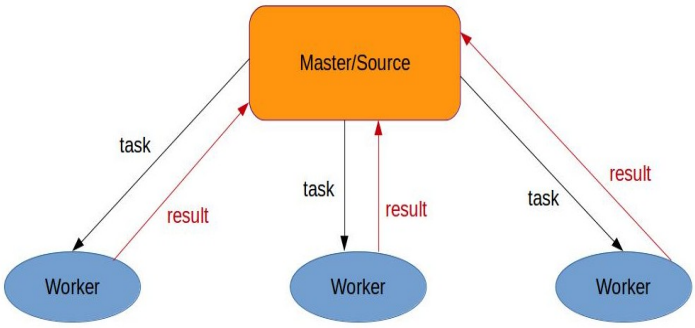
PRACE SoHPCProject ID

1610

# Smartphone Task Farm

*Anna Gradou*






The project application will be used at outreach events and science festivals to demonstrate how a task farm shares work across many independent nodes and how MPI applications distribute tasks across nodes that share information by message passing.

High performance computing clusters are used to solve problems that are too big for a single computer to solve efficiently. A cluster consists of computing servers which are connected together over a network and centrally coordinated specialised software. In this way, each server can contribute to the solution of a problem.

There are various techniques which can achieve this distributed environment. One of the most common ways is task farming where a problem is divided into a number of independent tasks. A master process is responsible for distributing the tasks among the available worker processes which send the results back to the master. The master process then combines all the results and produces the final result. In task farming there is no communication between the workers and can be only be applied to problems that can be split into independent pieces where every worker can process their tasks independently without exchanging information with other workers.

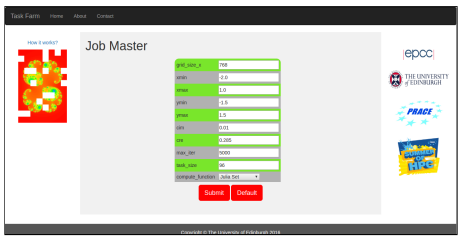



**Figure 1:** *Master Web Page Interface*

For some problems, task farming cannot be used and communication between workers is necessary. One such example is the Traffic Model that is used to predict traffic flow and to look for effects such as congestion. Transport model simulation is important as it can help better plan, design and operate transportation systems. In such models, each worker is responsible for the traffic upon part of the road assigned to it, but communication with other workers is required for the traffic entering or leaving from one road to another.

Two applications were selected to demonstrate these distributed computation techniques and participants could take part using their smartphone. The development of these applications took place at the Edinburgh Parallel Computing Center (EPCC).

## Task Farm Application

### Description

Participants can join a compute “cluster” and act as a compute node in a distributed task farm demonstration using their smartphone by loading a web page. The goal of the application is to present how tasks are shared across all the workers in a distributed application and how the final result is assembled from the pieces that each worker computes. Participants can see how their phones contribute small pieces of work that will be joined together to create the final result - which is a fractal image, on

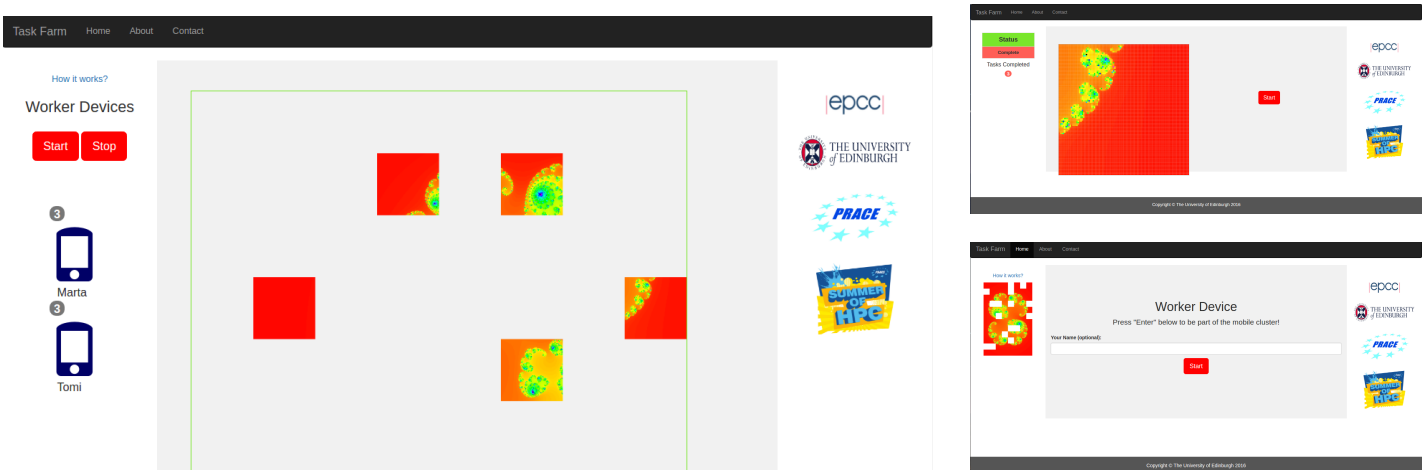



**Figure 1.2:** *The user interfaces: the Final Result Demonstration Interface, the Worker Interface and the Worker Registration Interface*

a large screen.

### What is a fractal

*"A fractal is a natural phenomenon or a mathematical set that exhibits a repeating pattern that displays at every scale."* as defined in Wikipedia. Fractals can be found in nature in trees, coastlines, clouds, seashells or hurricanes. The Mandelbrot Set and Julia Set fractals used in this application can be generated by a computer calculating a simple equation over and over.

### How it works

Every participants smartphone acts as a worker and there is only one master. At the beginning of the execution, the master chooses the parameters of the application. For example, the master can choose the algorithm that will be used (Mandelbrot Set or Julia set), the number of the iterations (how many times the equation will be calculated) or the size of every task (in pixels). An option of default parameters is also available.

After selection of parameters, the image to be generated is split into smaller pieces called tasks. Each task consists of the coordinates of a little square of the picture and a database stores all available tasks. Every time a user device loads the web page of the application and submits a form with a name, the device becomes part of the cluster and computes part of the fractal. A task is assigned to the worker device and removed from the list of the available tasks.

As soon as the worker is done, the calculated image is stored back to the database. The master collects these results and glues them together like a jigsaw puzzle. In this way, the users can see the image construction step by step as all the pieces will start gradually appearing on a large screen.

## Traffic Model Application

### Description

This application was developed to demonstrate the case when workers need to communicate. In this case the road is divided into a series of cells, either occupied or unoccupied. In each step, cars move forward if the space ahead of them is empty. Every user can control a different part of the road so for the boundary cases the users send messages to their neighbours to find out if the cell is occupied or not.

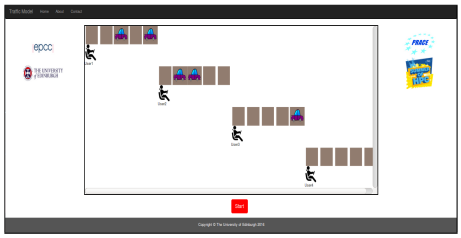


**Figure 2:** *Web Page Interface for the visualisation of the traffic*

### How it works

At the beginning of the execution the master user chooses the size of the road and the number of participants. The road is split into tasks which are stored in a database.

Every time a user registers by loading a web page and submitting a form, a part of the road is assigned to them. Each worker has to communicate with the boundary cases because they are not aware what is ahead from their point of view of the road. So they have to communicate and work together to solve the problem. The user can see on his screen how the cars are moving on the part of the road that he controls and the messages that he sends to the neighbours. On the master user screen, the traffic of the whole road and all the messages are demonstrated step by step.

### Development Tools

Both of the applications are client-server models with a RESTful web server as master that distributes tasks to clients running a HTML5/Javascript application. Flask microframework and Python was also used as well as MongoDB for the storage and the interaction of the information needed.

### PRACE SoHPCProject Title

Smartphone Task Farm

### PRACE SoHPCSite

EPCC, University of Edinburgh, UK

### PRACE SoHPCAuthors

Anna Gradou, [National and Kapodistrian University of Athens,] Greece

### PRACE SoHPCMentor

Amy Krause, EPCC, UK

Anna Gradou

### PRACE SoHPCContact

Catherine Inglis, EPCC
Phone: +44 (0) 131 651 3578
E-mail: c.inglis@epcc.ed.ac.uk

### PRACE SoHPCSoftware applied

HTML5,CSS,Javascript,jQuery,Python,RESTful web services

### PRACE SoHPCMore Information

www.w3schools.com

### PRACE SoHPCAcknowledgement

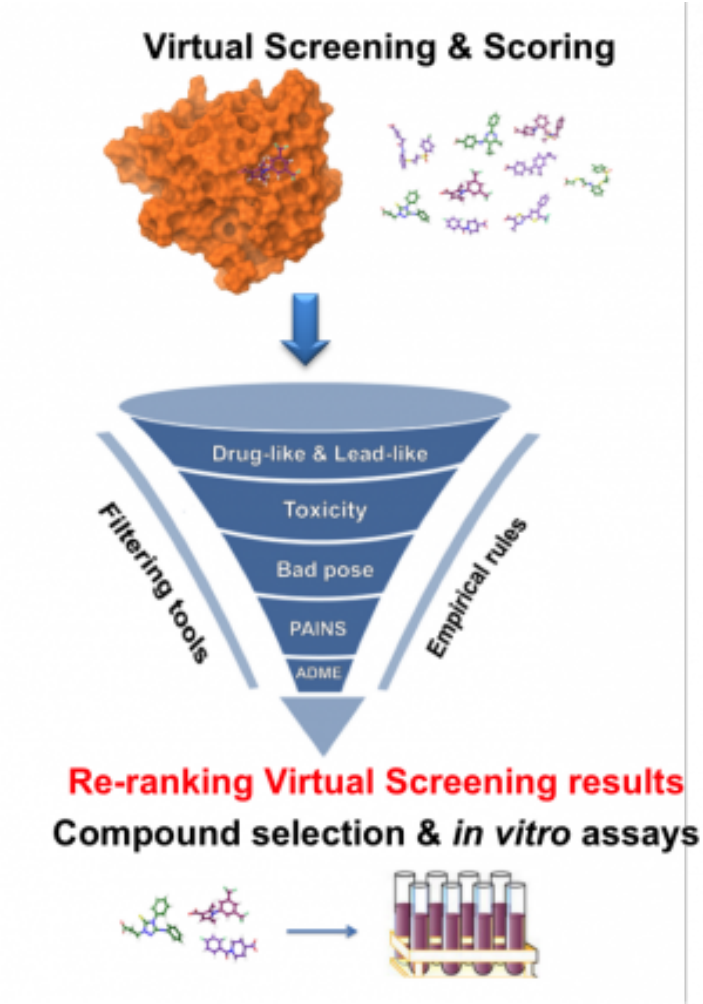
Special thanks to Dr Amy Krause for all the help and guidance.

### PRACE SoHPCProject ID

1611

# Re-ranking Virtual Screening Results

*Juan Eiros Zamora*




One of the initial steps in drug discovery projects is the virtual screening of small molecules against a set of targets to predict their affinity. A Python tool has been added to the ChemBioServer website to post process compounds arising from virtual screening to help enhance protein selectivity.

Despite technological improvements available to the pharmaceutical sector, the cost of commercialising a new drug doubles every 9 years.[1] One such technology is computational virtual screening, which aims to predict the structure of a target-ligand complex and the magnitude of the free energy of binding.

The systematic design of novel organic compounds is a very difficult problem, as it has been estimated that there are around $10^{60}$ synthetically-feasible molecules with drug-like properties. One of the initial stages in drug development is to explore this chemical space - fittingly coined as the Small Molecule Universe, using libraries which aim to capture its vastness with a small subset of very diverse molecules. Initially, thousands of putative drugs are 'virtually screened' against a desired target to predict their energy and site of interaction. This initial prediction is very important as the initial library of compounds is narrowed down to only the best scoring molecules which will later be analysed for further screening using more detailed computational models and experimental assays. This obviously allows for time and money to be saved.

One issue related to drug discovery is the problem of specificity. The bewildering complexity of a cell is still far beyond the reach of current simulation capabilities and the real targets of drugs are never isolated. Therefore, a compound that shows a strong affinity for a target could also have many off-target interactions, leading to undesired secondary effects. This is very common for protein families which are groups of evolutionarily related proteins which share structural similarities.

The objective of this project was to develop a tool that re-ranks virtual screening results based on screening a compound library against different protein members of the same family and selecting only those compounds that selectively score high for the protein of interest. Python was used as a programming language due to its ease of implementation and speed of prototyping, as well as for its rich ecosystem data analysis libraries. The program works as a command line tool, and has been added to the ChemBioServer,[2] a PHP-based web server that is hosted by the Biomedical Research Foundation Academy of Athens. The re-ranking program allows for different types of fil-

tering, with a default automated procedure that only needs a certain minimum of desired compounds to function. The compounds that pass the filtering are stored in Excel format for the convenience of a general user base.

## The reranking program

To start the initial development and testing of the *reranker.py* application, a docking data set of around 25000 molecules against 7 different protein structures of the kinase family (ALK1, ALK2, ALK5 and p38) was used. These virtual screening calculations were performed using Schrödinger's Glide docking software. The first step to load the data into Python was to convert it into the general readable CSV format using Maestro. Drug-like molecules often have multiple stereoisomers and tautomerization states that can contribute diversely to the overall binding energy to a particular target. Although the stereochemistry of a molecule is usually fixed, tautomerism is a dynamic process, meaning that a molecule can 'coexist' in multiple different states. During virtual screening, we treat these tautomers as different fixed molecules because chemical inter conversions are not considered in the calculation. Since the different tautomeric states are not represented through the naming conventions of molecules, it may appear as if there are several repeated entries for many compounds. To solve this, an additional post-processing step is needed. To uniquely identify each molecule, we have to parse their name into the SMILES format which accurately describes a chemical species using an ASCII string. An initial exploration of the kinase docking data set is shown in Figure 1 and is useful to understand what to expect out of the *reranker.py* algorithm. The predicted affinity of the drugs for each crystal structure follows a normal distribution centred around similar binding energies of -7 kcal·mol$^{-1}$. In this analysis, two different crystal structures of the same protein (ALK5) against a different one (p38) are compared. As can be observed, there is a higher degree of correlation between the two ALK5 crystals than there is for the p38 protein. Compounds ranking differently for different proteins can be exploited to find selective compounds.

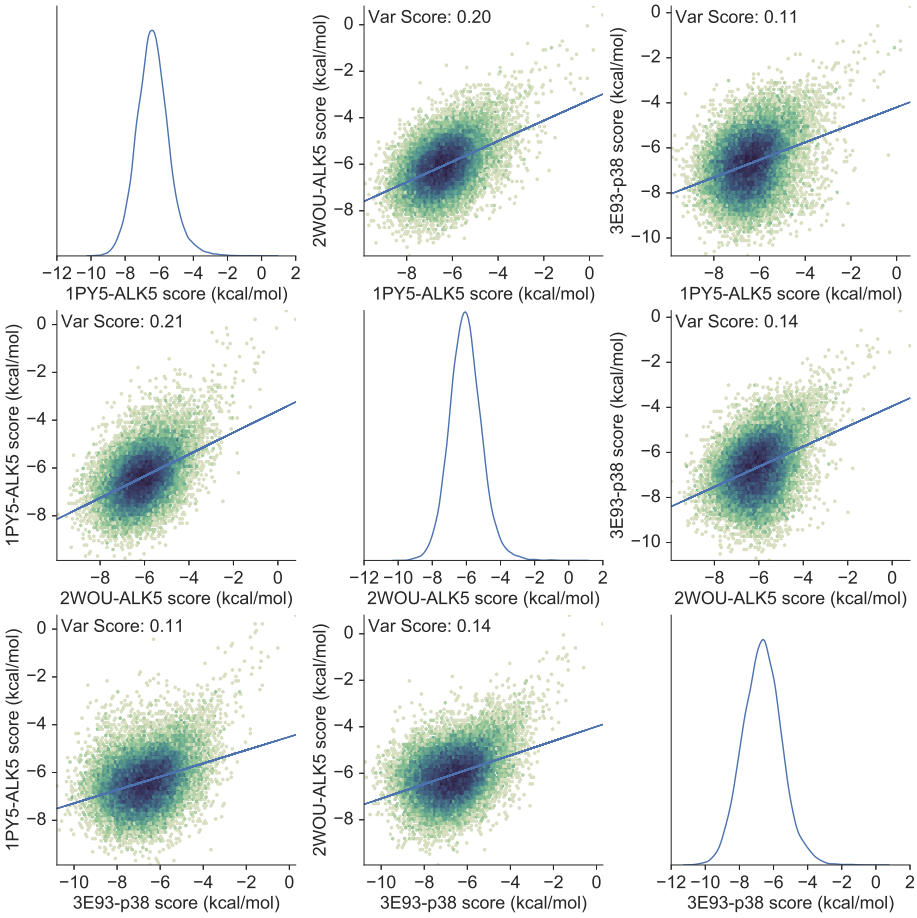



**Figure 1:** Overview of a virtual screening experiment of 25000 compounds on two crystals of the ALK5 kinase and one p38 crystal. Every dot on each scatter plot is a compound, with darker colours indicating a higher density.

Additionally, the results confirm what was expected that molecules generally retain their energy of interaction for a protein regardless of the crystal structure belonging to the same protein family they are docked on. The remaining 4 proteins in our data set are not shown here for simplicity's sake, but similar trends were observed, with the linear regression's explained variance scores ranging from 0.10 to 0.30. The idea behind the re-ranking algorithm is simple, the selective compounds will be those that have simultaneously low energies of binding for all the crystals of our target protein and high energies for the rest of the proteins that have been docked. To identify these compounds, we provide the user with three ways of defining a desired level of selectivity - automatic, manual or based on minimal desired energetic difference. In all three methods, the user has to specify the minimal number of compounds they want to retrieve from the re-ranking. The program heavily relies on the pandas Python package API, which provides all sorts of convenient functionality to work with data. The linchpin of this library is the Data Frame object, which allows the easy storage of data after reading it from CSV files. Panda's data frame objects support boolean indexing and have multiple vectorised methods which are faster than classic Python for loops. Since we observe that the distribution of docking energies can be expected to be normal, the automatic method starts by defining the cutoffs as the top 1% best scoring compounds for the target(s) and the top 1% worst scoring compounds for the rest of the proteins. We iteratively update the cutoffs by 1% steps until the minimum amount of compounds desired by the user meet the filter conditions. The manual method is more direct, as the user manually specifies the low and high energy cutoffs and a direct search is performed. The third method provides the user with an alternative way to define specificity. Often, the absolute values of the cutoffs might not be as important as the actual energy difference between the compounds for each protein. The larger this difference, the more selective the compounds will be. Therefore, the user can specify a desired level of energy difference and the pro-

gram will proceed in a similar fashion to the automatic procedure. It will start by defining the top 1% lowest scoring compounds for the target protein and the second cutoff will be set above the given score difference. Again, if no compounds match this filter, the low energy cutoff will be gradually increased by 1% steps, while the high energy cutoff will always be at least the desired amount of kcal·mol$^{-1}$ above it. These two methods are not guaranteed to succeed, as there might be no compounds that meet the selection criteria defined by the user. In such a case, the program falls back to the automatic method. After the filtered compounds are obtained in a data frame, we chose to output them as an Excel file which is available for download. This format was chosen to make it easier for scientists who might not be versed in programming to interact with ChemBioServer.

## Updating ChemBioServer

The *reranker.py* program was initially developed as a command line application, with the different methods of function and inputs being passed as arguments with flags. After local testing, we included it to the ChemBioServer web server. ChemBioserver is deployed using an Apache HTTP server with the back end written in PHP. For the integration of the program, the simple interface shown in Figure 2 has been developed. The user can upload multiple target files and multiple other docking results to filter against. One of the three methods can be chosen and corresponding input boxes appear dynamically using JavaScript. The input files are stored in the server and analysed by calling the Python *reranker.py* program through PHP after having parsed the appropriate commands. The results are stored for 24 hours and a link to download them is shown. The input CSV docking results are erased after their analysis to keep the server disk space as free as possible. We have successfully added the re-ranking functionality to the ChemBioServer and deployed it for general use. Based on the exploration of our docking data set, we have observed an expected amount of correlation of compound binding energies between the different proteins of the kinase family.

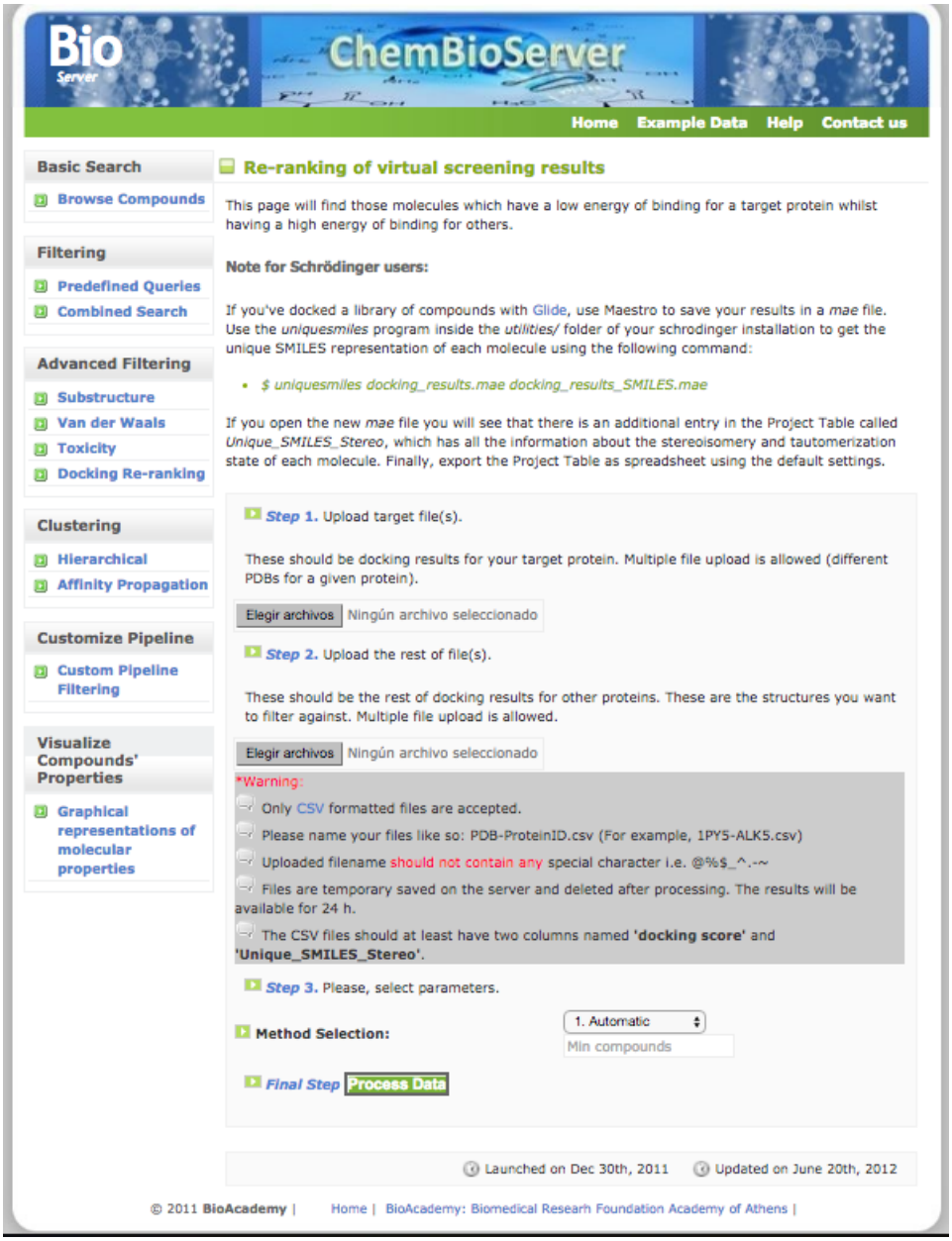



**Figure 2:** Interface of the docking re-ranking page in ChemBioServer.

## Conclusions

After extensive testing on the kinase data set we have found that the algorithm works best when used in its automatic settings, as user-defined cutoffs can usually be too stringent. Additionally, we note that the re-ranking procedure is a highly selective one, reducing a 25000 compound library to a dozen of potentially selective compounds. Nevertheless, we expect the degree of selectivity to be dependent to some extent on the protein family that is being studied.

## References


[1] Scannell, J. W. *et al.* (2012). Diagnosing the decline in pharmaceutical R&D efficiency. *Nat Rev Drug Discov.* 11(3):191–200.

[2] Athanasiadis, E. *et al.* (2012). ChemBioServer: a web-based pipeline for filtering, clustering and visualization of chemical compounds used in drug discovery. *Bioinformatics*. 28(22):3002–3003.


PRACE SoHPCProject Title

Re-ranking Virtual Screening results in computer-aided drug design

PRACE SoHPCSite

Biomedical Research Foundation Academy of Athens (BRFAA), Greece


Juan Eiros Zamora


PRACE SoHPCAuthors

Juan Eiros Zamora, Imperial College London, United Kingdom

PRACE SoHPCMentor

Dr. Zoe Cournia, BRFAA, Greece

PRACE SoHPCContact

Ioannis Liabotis, GRNET
Phone: +30 210 7474248
E-mail: iliaboti@grnet.gr


PRACE SoHPCSoftware applied

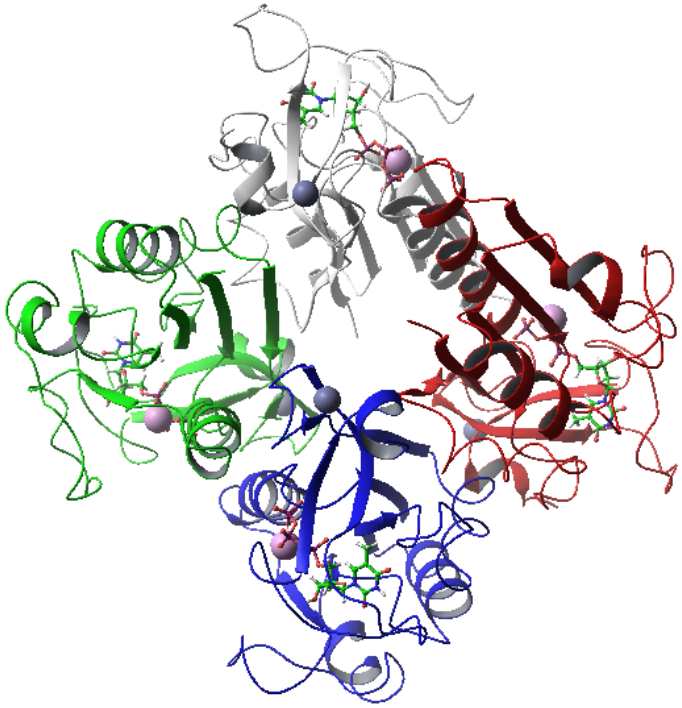
Python, PHP, Apache

PRACE SoHPCProject ID

1612

# Molecular Dynamics of hTK1

*Samanta Makurat*



hTK1 is a crucial enzyme for DNA metabolism, but the mechanistic details of its enzymatic reactions are still unknown Performing MD simulations is the first step towards understanding how the protein works.

The human cytosolic thymidine kinase 1 (hTK1) is a crucial enzyme for nucleotide metabolism. In native conditions it catalyzes the reaction of phosphate transfer from adenine triphosphate (ATP) to deoxythimidine (dT) nucleoside, forming deoxythymidine monophosphate (fig.1) which is the first step for further *in vivo* synthesis of DNA. hTK1 is found in nature in its tetrameric form and upon binding substrates its quaternary structure is significantly changed to adopt an open conformation that allows ATP binding.[1] In addition to its physiological role, hTK1 is also clinically important as it is required for the activation of nucleoside analogs such as radiosensitizing 5-bromodeoxyuridine or antiviral stavudine.

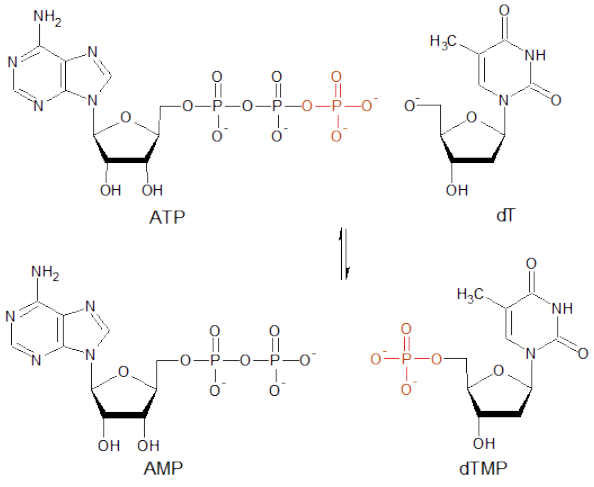



**Figure 1:** Phosphorylation reaction catalyzed by hTK1 in native conditions.

Despite its biological importance, the mechanistic details of the hTK1 enzymatic reactions have not been studied. Mixed Quantum Mechanics and Molecular Mechanics (QM/MM) simulations have become a powerful tool to study enzymatic reactions and can be used for this purpose. As a first step to prepare the QM/MM calculation, a realistic 3D structure of the enzyme has to be constructed based on structural biology studies. However, currently available hTK1 structures are hampered by different issues: (A) There is no hTK1 structure in the activated (open form) complexed with either substrates or products of the reaction (only ones complexed with deoxythymidine triphoshphate, TTP at the phosphoryl acceptor binding site), (B) The active site of hTK1 is very flexible and only one of four monomers (for only one of 3 available structures[2,3,4]) includes the full structure of the active site.

## Aim

The aim of this project was to model the hTK1 in both its inactive and active forms using Molecular Dynamics (MD) simulations and to observe significant structural changes upon binding the inhibitor that occupies both phosphoryl acceptor and donor binding sites, ($P^1$-(5'-adenosyl)$P^4$-(5'-(2'-deoxythymidyl))tetraphosphate, 4TA), as the literature suggests.[4] Transition from a closed, inactive conformation to an open, catalytic state is crucial, providing space for the adenosine moiety. Information gained from these simulations will also be used in setting up the QM/MM calculation to

monitor the phosphorylation of thymidine and other substrates, such as bromodeoxyuridine and other analogues that are used as radiosensitizers.

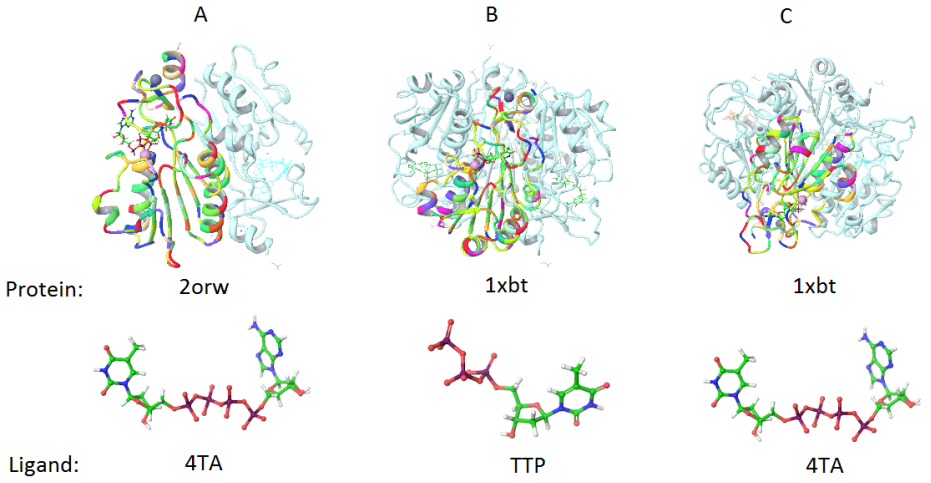



**Figure 2:** Simulated structures. The two control systems were A - dimeric, open form tmTK of PDB code 2orw with its cocrystallized inhibitor as a ligand, B - closed, tetrameric hTK with its cocrystallized inhibitor (PDB code 1xbt) and C - hTK (1xbt) but with ligand taken from tmTK, requiring the structure change upon binding.

## Methodology

### Model construction

The activated (open) form of the hTK1-like enzyme containing inhibitor mimicking proper substrates (4TA) from ***Thermotoga Maritima*** (tm) TK was used as a template and a first control system (system A, fig. 2).[4] The tetrameric hTK1 in its inactive form, which has been co-crystallized with an inhibitor (TTP) blocking the open conformation , was used as a second control simulation (system B, fig. 2). Finally, the active-form ligands (4TA) were imported (either by docking or overlapping structures) in the tetrameric hTK1 closed form in order to monitor changes upon substrate binding (system C, fig. 2, further referred to as C1 for overlapped docked and C2 for docked structures). For all the systems, the full structure of the active site was completed by homology modeling (from PDB code:1w4r) in the monomers that lacked it. Other missing fragments, located at N- and C-terminal ends have regulatory roles and are not required for catalytic function of the enzyme.[5] All these steps were performed with the use of Maestro (Schrodinger) software.

### MD simulations

The MD trajectories for all 4 systems were generated with GROMACS, using the AMBER99SB*-ILDN all atom force field and the TIP3 water model, with $Na^+$ and $Cl^-$ ions added to neutralize systems and mimic physiological conditions (150mM). All proteins were solvated into a cubic box large enough to ensure separation of the protein from its periodic image. The ligand topology files were prepared with Acpype (link), and the charges calculated with Gaussian09 (link) DFT/B3LYP calculations, 6-31++G(d,p) basis set.

## Results

### Systems Equilibration and Stability

In order to monitor structural variations and overall indication of the simulation stability for each system, the RMSD of the C$\alpha$ atoms for all four simulations was calculated (fig. 3). We observe that for all simulations a plateau is reached at about 40 ns, and therefore the trajectories that were further considered as production runs and used for analysis are 460 ns long (40-500 ns). Visualisation of the trajectories provided us additional observations. We observed that during the simulation of the C2 system, one of the ligands leaves its pocket. That observation, along with ligand-protein interaction analyses performed for remaining 3 monomers, suggests that the docking methodology was either inadequate for this system or was performed incorrectly. Therefore this system will not be discussed further and more emphasis will be given on system C1, which placed the ligand in the protein by overlapping crystal structures 1xbt and 2orw.

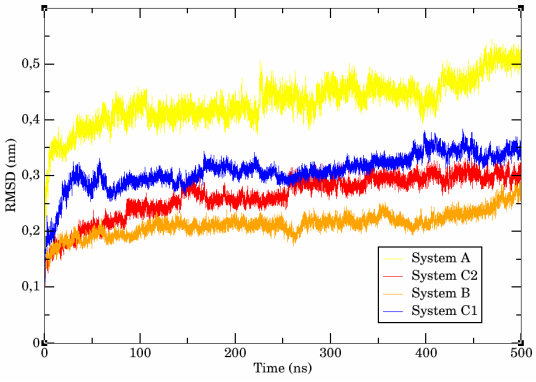



**Figure 3:** RMSD of the backbone atoms for all the tested systems. Notice, that system A is a dimer, while all the others are tetramers and should not be compared with them.

### The conformational changes

As mentioned above, binding on the phosphate-donor site (ATP) requires a conformational change in the quaternary structure to an open state. In tmTK we observe that the adenosine moiety is positioned at the interface of two monomers and sandwiched between Tyr13 and Leu29 of different monomers, which are 11 Å, apart.[4] These aminoacids are equivalent to Phe29 and Ile45 in hTK1, reported to be only 7.5 Å apart in the closed conformation, as reported by the same study. Indeed, after checking distances between these two residues in System B (containing TTP, closed form) and System C1 (where we expect the opening upon ATP binding), we observed a significant difference. During the course of simulation, the distance between these residues oscillates between 7 Å for the closed conformation, and 13 Å for the system bound with ATP.

Another region that changes upon ATP binding, as reported in the literature, is a flexible loop of 18 aminoacids (see Fig. 4, dark blue and yellow), demonstrating a more rigid structure than the one without the substrate.[1] This region is changing in our structure as shown in Fig. 4, although more calculation time is needed to assess whether this change is statistically significant. Regardless, in order to measure the change, RMSD for this short part of the protein has been performed. Analysis of the results (Fig. 5) allows us to conclude that in contrast

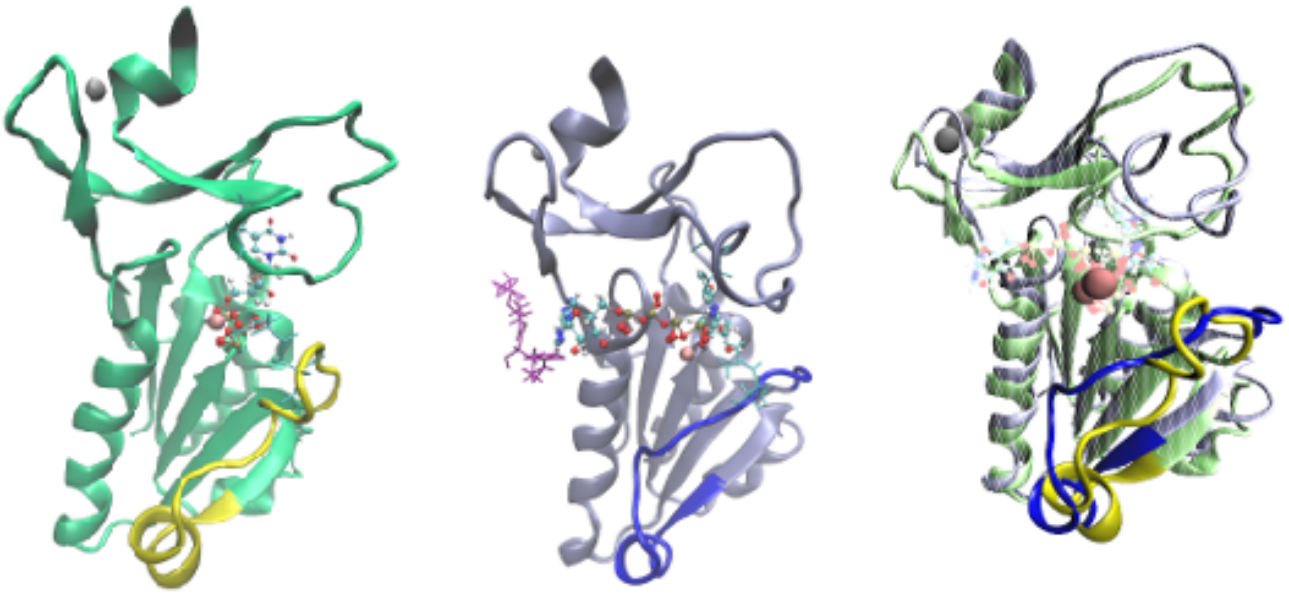


**Figure 4:** Overlay of system B (green) and C1 (blue). One monomer was chosen from both tetramers for better representation.The flexible loop (shown as yellow and dark-blue respectively) shows the change of the structure during the simulation. Moreover the hydrophobic pocket arginine residues from neighbouring monomer are shown (pink) for system C1.

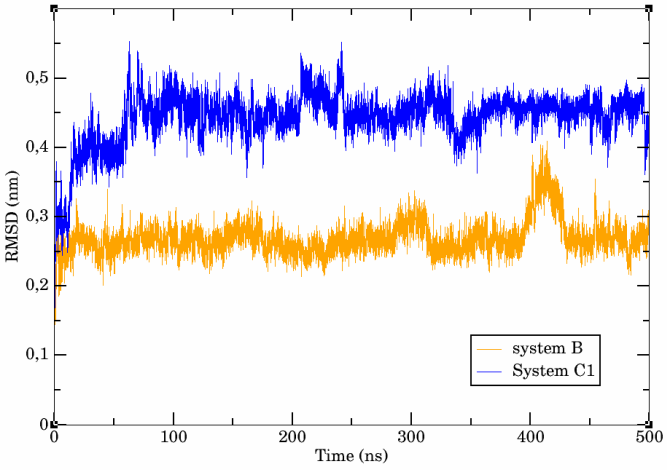



**Figure 5:** RMSD of flexible loop in chosen monomers of B and C1.

to the control simulation (tetramer without ATP), the loop is changing firmly, especially during the first 150ns of simulation, later becoming more and more stable, but it still acquires quite a different structure compared to the initial crystal structure. The high RMSD values observed for the "opening" system C1 and not for the stable closed B suggests that it might be changing to the desired conformation.

## Discussion and Conclusion

MD simulations of hTK1 complexed with inhibitor both at the phosphoryl acceptor and donor sites was a chal-lenging task and longer simulations and replicas will be needed in order to validate the results and better understand the processes related to binding the substrates. Still, comparing to other simulated systems and available literature we can conclude that the goal of the project was more than achieved - the structure shows significant changes upon binding the ATP. Also, the preliminary studies of the binding pocket (not shown here) shows its validity. Especially monomer B of C2 structure, where ATP is nicely placed in the hydrophobic pocket, just as suggested in the literature. I believe, that the representative structure obtained from this simulations is enough to perform QM/MM calculations. Before that, further analyses will be needed - especially protein-ligand interactions studies (including identification of catalytic bases), in depth analysis of binding sites and choosing the representative structure.

## References


[1] Segura-Pena, Dario, et al. "Quaternary structure change as a mechanism for the regulation of thymidine kinase 1-like enzymes." Structure 15.12 (2007): 1555-1566.

[2] Welin, Martin, et al. "Structures of thymidine kinase 1 of human and mycoplasmic origin." Proceedings of the National Academy of Sciences 101.52 (2004): 17970-17975.

[3] Birringer, Markus S., et al. "Structure of a type II thymidine kinase with bound dTTP." FEBS letters 579.6 (2005): 1376-1382.

[4] Segura-Pena, Dario, et al. "Binding of ATP to TK1-like enzymes is associated with a conformational change in the quaternary structure." Journal of molecular biology 369.1 (2007): 129-141.

[5] Birringer, Markus S., et al. "High-level expression and purification of human thymidine kinase 1: quaternary structure, stability, and kinetics." Protein expression and purification 47.2 (2006): 506-515.


PRACE SoHPCProject Title

Molecular Dynamics Simulations on Human Thymidine Kinase 1

PRACE SoHPCSite

Biomedical Research Foundation, Academy of Athens, Greece

Samanta Makurat

PRACE SoHPCAuthor


Samanta Makurat, University of Gdansk, Poland


PRACE SoHPCMentor

Zoe Cournia, BRFAA, Greece

PRACE SoHPCSoftware applied

Maestro, Gromacs

PRACE SoHPCAcknowledgements


The calculations were performed with the use of ARIS supercomputer in Greek Research and Technology Network. Apart from my supervisor Dr Zoe Cournia, I would also like to thank Dr Dimitris Dellis (GRNET) and Giannis Galdadas (BRFAA) for all the training provided, support and insightful discussions.


PRACE SoHPCProject ID

1613

# Performance Dashboard

*Thomas Wright*


Is your super important scientific simulation running super slow, even on a supercomputer? Then the performance analytics dashboard can help, giving you an in-depth view of your application's performance.

In the era of supercomputers, scientists and businesses are able to use these machines to process large amounts of data, all with the goal of helping them to model and better understand the world. From simulating the brain, exploring the inner working of stars, to compiling and indexing all of human knowledge, supercomputers aim to provide some of the the greatest scientific breakthroughs of our time. Efficiently utilising all of this power is by itself very challenging. Whilst most applications struggle to efficiently use the two or four cores provided in desktop computers, in order to benefit from a supercomputer, one must efficiently use hundreds, even thousands of cores at the same time.

This project aims to make these challenges more manageable by developing tools to monitor the performance of parallel programs on Fionn - Ireland's largest supercomputer. This will involve developing a web application which visualises all of the resources a running job uses on a supercomputer, while at the same time identifying underutilised nodes and any possible bottlenecks which prevent a job from scaling.

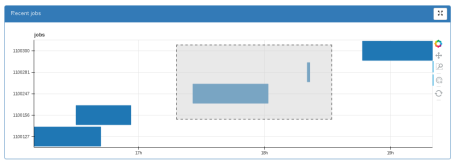
## The project

Currently when a user wants to investigate the performance of their application there are a number of tools available, ranging from command line utilities to dashboards which monitor the various allocated nodes. However, even with these tools, it is difficult to answer a simple questions such as "Am I effectively using all of the nodes I have been allocated?", "What are the bottlenecks in my application?", and "What does my application's performance look like over the duration of its execution?". The goal of this project was to build a single web application which allows users to look at their past jobs and answer all of these questions through the use of performance profile visualisations.

The main challenge facing this project was how to process all of the existing performance and job data collected on Fionn so these could be presented in a web application. Out of the wide range of available languages designed for data processing and analysis, it was decided to use the Python programming language as it has a large repository of open source packages which help with many aspects of building data driven web applications.

The data from Fionn comes in the form of XML files exported from the Ganglia monitoring system and from Torque task scheduler text logs. Processing these files we were able to store the most important information in a PostgreSQL database. The files were fairly easy to process, however a supercomputer such as Fionn produces tens of Gigabytes of data a day so it is essential that the performance dashboard can handle this volume of data in a timely manner.

Most of the work involves the processing of Ganglia files as these include readings for many different metrics taken every second for each of Fionn's 320 nodes. This is handled using custom import routines such as the lxml XML parser library and PostgreSQL's COPY statement for bulk data imports.



With the data imported, we get on to the fun bit – presenting and visualising the data in a web application. There are many different frameworks available for building web applications in Python, but I chose to build the performance dashboard using Flask, as it is simple and modular and allows for the use of a different database system should future scaling require this. When implementing the user interface I first focused on providing basic building blocks for displaying plots and collecting these into dashboards, out of which I then implemented the UI. This structure greatly simplified the design of the application and will make it easy for it to be extended with new types of visualisation and new performance metrics in the future.

A core feature of the web application is to go beyond producing static plots of data and produce interactive visualisations using dynamic plotting. These allows the user to pan and zoom the data in their browser. This allows for the visualisations to serve as an essential part of the application's user interface. For example, we can show the user a timeline of their jobs and allow them to hover over a job for more information, or click on a job to open an in-depth profile of the job's execution. The Bokeh Python library wa used to make this kind of interactive visualisation possible.

## Results

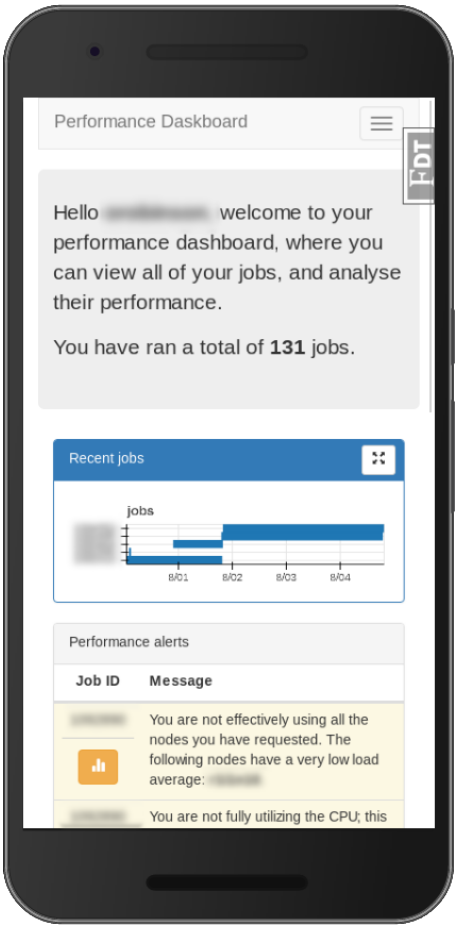
In just 6 weeks it was possible to combine the latest web technologies and the data collected from Fionn to build an interactive performance analysis dashboard. This allowed to present a clear view of a user's activity on Fionn and allowed them to select individual jobs for greater in-depth performance analysis. As it is not always easy for users to interpret graphs, we have also implemented automated analysis which detects common performance problems and explains these issues to users. As an example, users often try to speed up applications by running it upon more nodes. However, unless their application is able to scale to the number of allocated nodes, computing resources will be wasted. This is detected by checking that each requested node is actually active.


The most important page of the application is the job performance dashboard which presents a full execution profile of a user's job using visual time lines of various performance metrics. This has proved to be very useful in understanding the performance characteristics of jobs and revealing potential for performance. For example, a common design for jobs is to spend most of their time executing CPU intensive computations but periodically to stop and write their results to disk. Depending on how efficiently this writing is done and how they distribute it across the various nodes, it can easily become the limiting factor in a job's performance. To make matters worse, it is often difficult to understand how great an impact this can have and where the room for improvement is. However, using our application, it becomes easy to assess this impact by comparing the CPU utilisation, and the CPU I/O wait time. Combined together this shows the proportion of the time the CPU is blocked from performing computations by I/O operations. If the user determines that this significantly impacts the performance of their job, one way to improve it is by storing more data in fast system memory before writing it to disk. However, one has to be careful to avoid exhausting the limited supply of available memory. Taking this into account we display a job's memory usage and thus this is easy to keep track of. Using visual performance presentations proved very effective as it makes it possible to get an idea on the behaviour of an application as well as providing a clear visual indication of differences in behaviour between applications or between the nodes executing a single job. This allows for optimisations in code and to be identified.

Key to the effectiveness of the application is its design as an interactive web application which allows users to access the tool using a web browser. By developing the performance dashboard on web technologies, it was possible to develop a single application which works on devices ranging from mobiles to desktop computers. The user interface utilises responsive design so elements rescale and the layout adapts to the resolution of the device being used.

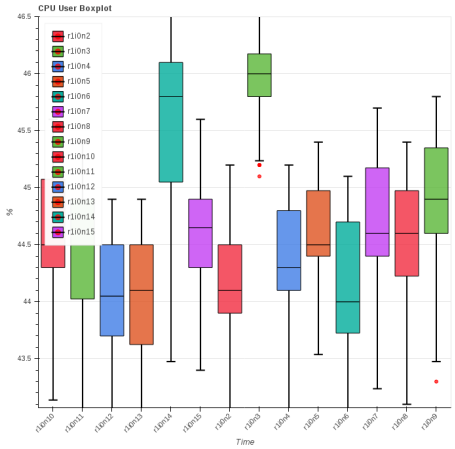
Over the course of the project, I realised how rapidly web technologies have evolved and how much easier it has become to build compelling user interfaces. Whereas in the past graphics rendering would have to be performed by a server, due to the increasing power of Javascript and technologies such as AJAX and Websockets, it is now easy to get the browser to do more of the heavy lifting. This improves page load times and interactivity, making websites seem less static and more like full blown applications. By moving to the web, applications can reach a much larger audiences as they can run on all platforms and users do not require the installation

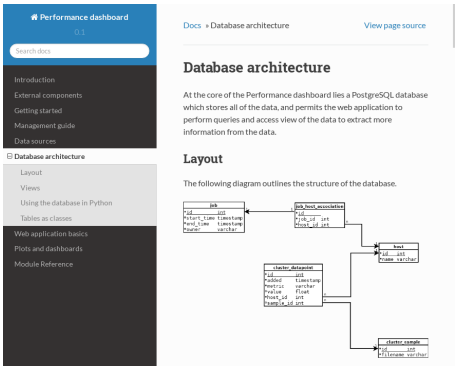
of any software. As web technologies become more powerful, I believe that other scientific monitoring applications could also benefit from this approach in the future.




Whilst the application is at a stage where it can be useful for understanding the performance of jobs, more work will need to be done before it can be placed in the hands of users. This includes adding a mechanism to transfer the datafiles which power the application from the secure node on Fionn to the web application and deploying the application to an internal server. There is also much scope to extend the application and to perform a wider range of analysis.

Whilst we currently only use simple checks to detect common performance issues using SQL queries, it would be possible to extract a lot more information, such as long term performance trends and comparative analysis of different runs of jobs, using big data analysis techniques. It would also be easy to extend the application to utilise other data sources such as I/O profiling and power efficiency monitoring. In order to enable the continued development of the application, I have adopted a flexible design and provided extensive documentation of the application to get future developers started as quickly as possible. Hopefully with this work, the performance dashboard will be able to expand into a comprehensive performance analysis tool for users across Fionn.




PRACE SoHPCProject Title

Development of a Performance Analytics Dashboard

PRACE SoHPCSite

ICHEC, Ireland

PRACE SoHPCAuthors

Thomas Wright, United Kingdom

Thomas Wright

PRACE SoHPCMentor

Servesh Muralidharan, ICHEC, Ireland

PRACE SoHPCThanks

Simon Wong, ICHEC, Ireland
Eoin McHugh, ICHEC, Ireland

PRACE SoHPCContact

Thomas Wright, University of Edinburgh
Phone: +44 713 254 795
E-mail: t.d.wright@sms.ed.ac.uk

PRACE SoHPCSoftware applied

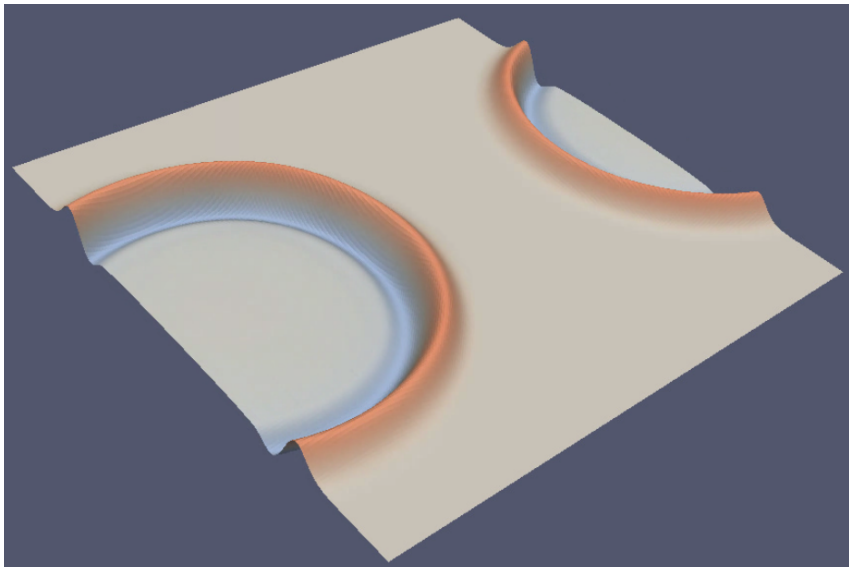
Python, Flask, Bokeh, PostgreSQL

PRACE SoHPCProject ID

1614

# Shallow water equations

*Jiri Blahos*



The aim of this project is the simulation of fluid behaviour, namely shallow water behaviour. Such models can be very useful while studying various natural phenomena, such as ocean waves - including tsunami waves, or atmospheric air flows.

The shallow water equations are a set of mathematical equations, which describe the physical behaviour of fluids (not only water). As suggested by the name, they simulate the behaviour of shallow waters which describes situations where the horizontal span of the simulated area is far larger than the vertical one and where the vertical velocity of the fluid does not vary with depth (or the variations are negligible). The word *shallow* can be a bit misleading here as shallow water equations are commonly used to simulate tsunami waves in the ocean. Even though the ocean is not shallow (in the common understanding of the word), its surface area is still far greater than its depth. Shallow water equations - as other equations, are solved using a computers which given specific input, evaluate the equations and provide the behaviour of the system as an output. This output is typically in form of set of numbers, which describe height, velocity pressure and other physical parameters present in the simulation.

## 1 Project assignment

As my starting point, I was assigned to work on previous Fortran code which simulates shallow water equations. I began by studying and understanding this code. Even though it was well written, it is never easy to read someone else's code - especially given that Fortran was something new for me. Given I am mostly program in C# and C++, the syntax of Fortran appeared unnecessarily complicated at certain points. Besides this, I saw it as a good experience to learn a new programming language.

My project was titled "Visualisation of fluids and waves" which allowed for plenty of freedom in how I would direct my work. Furthermore, my project supervisor, Dr. Adam Ralph explained how the final result should ideally work and allowed for flexibility on how things could be done. The goal of the project was split in two parts - code parallelisation and output visualisation. For the parallelisation part the goal was simple - alter the code so it can run on supercomputers (namely *Fionn*, the Irish most powerful supercomputer) and run faster. The visualisation included the challenge of a 3D graphical view of the shallow water model output and ideally for this to be created as the output is produced by the model in "real time".

It is always good to know the final goal of a project - especially when it comes to programming, to know *how it should look like* and *what it should do*. The question of *how it should be done* arises as secondary to this and of course, my project supervisor was of great help here. We discussed the problem, the pros and cons of different approaches and adjusted the plan relative to the current situation.

# 2 Project elaboration

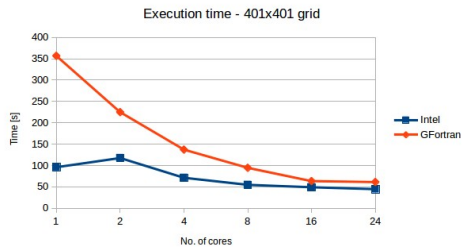
## 2.1 Parallelisation

One of the crucial points when it comes to physical simulations is time. It is easy to create a detailed mathematical model, which would require a long time to evaluate on a computer. The standard way of writing a computer program is simply putting down instructions which are then executed one by one. But what if there are instructions which can be executed independent on each other? Could we execute them at the same time, in a parallel manner? That is exactly what parallelisation is - taking advantage of a current multiprocessor computer architectures and writing our code in a clever way so we can execute as many instructions in parallel as possible. Simply running any code on a supercomputer wont help much. It needs to be properly adjusted.

I decided to use OpenMP for parallelisation of the original Fortran code. OpenMP is a software library which allows for quite simple and straightforward code parallelisation by using threads with shared memory. Given than Fionn's computational nodes each offer 24 cores with shared memory, the choice of OpenMP was correct. The availability of 24 cores allows for the possibility of splitting work into up to 24 independent "threads". This is good enough for the shallow water simulation I was working on, which is not "big enough" to require higher degree of parallelisation.

Of course, code parallelisation has its obstacles. You can't just let your workers (threads) do whatever they want, and at any time. Sometimes all workers must wait for each other and synchronise as some tasks can be performed by just one worker. When such restrictions are not carefully taken into account a program can still run and produce chaotic and unexpected results - and it is very difficult to identify the reason for this.

This is how the first part of my project looked like. Using OpenMP, modifying the simulation code, debugging it, trying to make it efficient and more parallel.





## 2.2 Benchmarking

After completing the parallelisation part, some code benchmarking was carried out. This is important so as to highlight any potential benefits from parallelisation. As expected, the parallelised code run faster. The more computing cores used, the faster the simulation ran. However, even when using 24 cores, the code ran only 5 times faster in the best case scenario (see attached table). This was a bit of disappointment. Reasons for this could be that the simulation is still not "big enough", or that it is not capable of running faster in it's nature, or that I wasn't able to find the optimal way of parallelising the code. Despite this, the goal was still met - the code run faster after being parallelised.

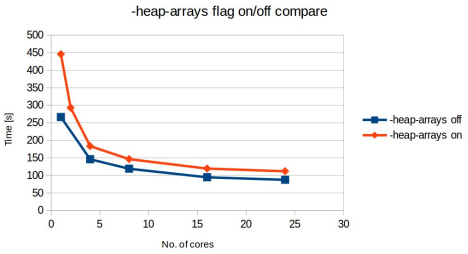
## 2.3 Visualisation

The next part of the project was visualisation and what I was looking forward to most. Visualisation is about graphics, pictures, animations, videos and how starting with a huge pile of numbers this can be turned into a nice looking image which everyone can understand. That for me is the goal of visualisation - to present data in a way people can understand and see the overall result.

After discussion with my supervisor, ParaView was chosen as the visualisation tool to use. ParaView is a free

distributed software used for visualising scientific data. It offers plenty of built in mechanisms for loading data in various formats, manipulating them and viewing them, in both 2D and 3D. My simulation code had to be altered slightly so its output could be in a format compatible with ParaView. After that, I could experiment with visualising the data. The simulation code output consists of a height field - describing the height of fluid surface at each point of space, and a velocity field - which describes the velocity of fluid at a given point. To visualise the height field, I let ParaView render a 3D surface based on height. I could then stretch this surface to make the fluid waves appear bigger and more visible and colour it to mark out high and low points of the surface. After using transparency, Fresnel effects and experimenting with different colours, the surface began to look similar to a real water surface.

As previously stated, another goal was to make a "real time" visualisation - to display new model output as it is produced. This was achieved by writing a simple ParaView script which makes it look for new output files and adds them to the visualisation collection. Using this script, ParaView is able to display the simulation progress and add new frames to the animation when new output is produced. It is not an ideal solution though, since running this script makes ParaView non-responsive to certain user actions and the only way to stop the script is to shut down ParaView completely.




## 2.4 Custom OpenGL app

After experimenting with ParaView and producing some nice images and videos of shallow water simulation, there was still some time left. Therefore, I decided to write my own visualisation application in the hope of improving the quality of the visualisation result. I already had experience with OpenGL, so I used one of my previous applications and modified it for the purpose of fluid rendering. There wasn't too much time left, but I still managed to render some decent images, as can be seen in the pictures. Furthermore the application is capable of loading model output in "real time" as data comes in and is able display it in the form of an animation.

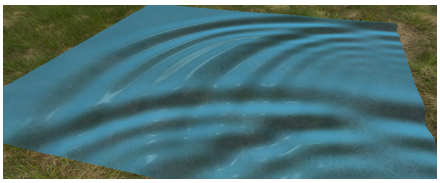
## 3 Project summary

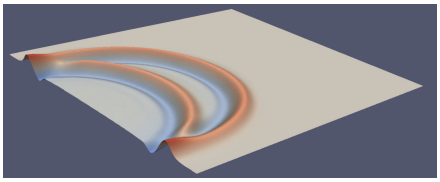
As my work on this project has completed, its time to look back and compare my initial goals to what I actually managed to achieve.

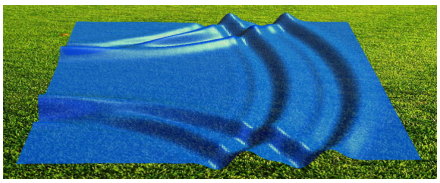
For the parallelisation part, I managed to modify the simulation code successfully and test it on Fionn. The results were not too overwhelming as the performance gain could probably be better. Despite this, I managed to produce a stable parallel code, which demonstrates the methods of parallelisation and still allows to speed up the simulation computations. So I would consider that the primary goal in this part was met.

In the visualisation part of my project things went quite well. The images created in ParaView look really nice even though things can always be improved. Working on the custom OpenGL application turned out to offer plenty of opportunities, but there wasn't enough time to make more of it.

To summarise the project, I learned plenty of new things. New programming techniques, new software, new approaches. I also came across some interesting bugs, which again extended my knowledgebase about code debugging and analysis. Therefore, I see this project as a really valuable experience which I would recommend to everyone.







## 4 Acknowledgments

I would like to thank all the people who supported me during my Summer of HPC work. First of all, my project supervisor, Dr. Adam Ralph, for his guidance, plenty of good advice, and for being always kind and patient to me. I would also like to appreciate the support from all the ICHEC staff, especially Simon Wong and Adam Murphy, for their support in administrative parts of the project. Special thanks also goes to Dr. Enda O'Brien, who created the original shallow water simulation code, which I used during my project.

Next, I would like to thank IT4Innovations, for supporting me in this project. Namely I thank Ing. Tomas Karasek, Ph.D. and Ing. Karina Pesatová, MBA, for their support and help with the application for this project.

PRACE SoHPCProject Title
Visualisation of fluids and waves

PRACE SoHPCSite
ICHEC, Ireland

PRACE SoHPCAuthors
Jiri Blahos, Czech Republic

PRACE SoHPCMentor
Dr. Adam Ralph, ICHEC, Ireland


Jiri Blahos

PRACE SoHPCContact
Jiri, Blahos, VSB - TU Ostrava
Phone: +420 605 051 249
E-mail: j.blahos@seznam.cz

PRACE SoHPCSoftware applied

PRACE SoHPCMore Information

PRACE SoHPCProject ID
1615

# Virtual Reality exploration

*Alejandro Rodriguez Segrelles*

Studying and navigating through 3D generated parts of the human body is not just for doctors but for everyone, as long as you have a smartphone and Virtual Reality glasses

The goal of this project was to create an interactive game that could allow the player to navigate and contemplate formed human parts from different angles.
The game engine Unity3d was used as this makes the process of creating a game easier than just programming in notepad.
The medical data that was used came in Digital Imaging and Communications in Medicine (DICOM) format which consists of many 2D scans piled up on top of each other.
For this project, Computed Tomographies (CT) were used as input for the medical data, but it is also possible to use other technologies (Magnetic Resonance Imaging, radiographies, ultrasonographies, etc).
With this DICOM data, a 3D model is created. With this format one can identify different tissues, so tissues which will be studied will be isolated from the rest to facilitate its visualization.
All this sounds technical and specific, but the objective is to make it simple for the end user, with the player being able to move and look around a human body with Virtual Reality.

**Figure 1:** CT scans from all 3 axis

## Obtaining the mesh

According to Wikipedia, *a 3D mesh is a collection of vertices, edges and faces that defines the shape of a polyhedron*.
Now that we know what is a mesh, the question is “How do we make it?”
Doing it by hand would be an inaccurate and slow process, that is why CT scans are used. But just using these scans will not magically get the desired results.
IT4Innovations uses Blender for all 3D modeling related tasks, because it is free and open-source. This allowed them to program their own plugin to be able to work with the DICOM format in a feasible way by allowing for the processing of large medical data on supercomputer nodes. The plugin is still not ready for general release, but its current function has been useful in this project.
Once Blender is executed in the supercomputer, it is time to start. The plugin is able to load a DICOMDIR file which stores information about all DICOM files.
From this point until the very end it makes a huge difference to use a supercomputer instead of a normal computer as processing these files requires less time.

Once DICOM files are loaded, one can start viewing the scans one by one in the Blender *2D image viewer*. For smoother visualisations, the sagittal (left side to right side) and coronal (front side to back side) from the axial (top side to bottom side) view slices were generated, as you can see in *Figure 1*.
We can then isolate a desired tissue for closer inspection once the mesh is generated.
We have plenty of tools and functions for that purpose available, for example:

- Bright/contrast sliders
- Cut image
- Blur image
- Threshold image values
- K-means clustering
- Flooding of areas
- Create boundaries
- Poisson reconstruction

Most of them are self-explanatory and some - such as K-means, can ease and make clearer the process of image segmentation. We briefly explain how K-means works. The scans are in gray scale thus for each pixel, the red, green and blue values are equal. K-means is a mechanism for coupling and is able to identify different groups from similar *gray* pixels value. These sets can then be "painted" in different colours in a *2D viewer*. The more groups which are set can allow for more details to be identified - as can be seen in Figure 2.

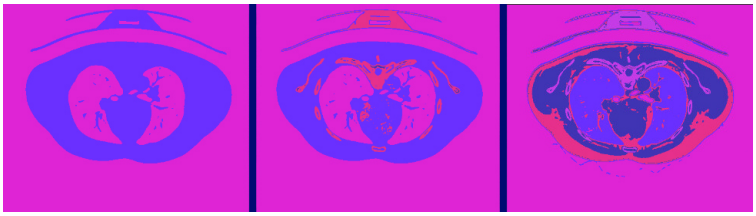


**Figure 2:** Scans with 2, 4 and 6 K-means groups

The coupling itself does not achieve anything, but it does provide a clearer visual image identifying which tissues we can select to isolate. After this, it is time to plant a seed. To accomplish this, we will point to a specific pixel which will select the group the pixel is assigned to. We can then flood to the selected seed, meaning that every group will be deleted but the group in which the seed belongs to.

Every scan that was used has some selected areas, but it is a 2D surface and we just care about its edges. Because of this, the tool *create boundaries* is the most appropriate to use as it detects edges from every scan and leaves CT

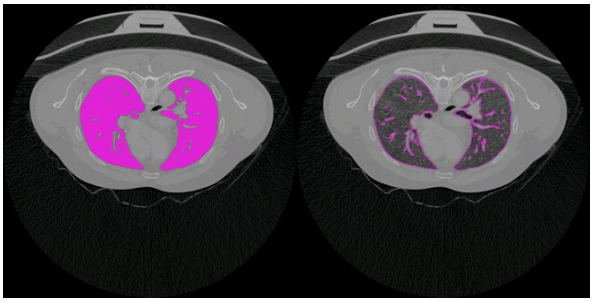


**Figure 3:** Tissue flooded in the left, flood boundaries created in the right

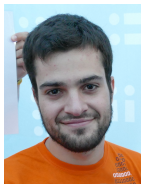
Once a piece of tissue has been chosen, it can then be transformed into a 3D model. Since the selected pieces also include the inside part of tissue, applying the *create boundaries* function will detect the edges from the scans. All that remains is to generate the mesh which is achieved using the *poisson reconstruction* method. Once ready, the mesh should appear in the 3D viewer used. Some final enhancements such as smoothing the shape can be carried out to make it easier for the eyes to view. It can also be exported to be later used in different software.

## Making the game

With the 3D model ready, its time to open Unity. Virtual Reality was one of the planned features of this game and thankfully, Google released a Software Development Kit for Unity that makes the game compatible with Virtual Reality visualisation devices through a very simple configuration process.
Unity is a complete game engine, therefore it is easy to make assets due to its compatibility with a wide variety of formats. Because of this, importing previously obtained 3D models wasn't difficult at all.

To make a predefined player route we used splines (curves). We set a high number of waypoints for a more precise journey and thanks to the spline scripts, the transition from one waypoint to another was smooth, delivering a pleasant experience to the user.

PRACE SoHPCProject Title
Journey to the centre of the human body



Alejandro Rodriguez Segrelles

PRACE SoHPCSite
Ostrava, Czech Republic

PRACE SoHPCAuthors
Alejandro Rodriguez Segrelles, Spain

PRACE SoHPCMentor
Milan Jaroš, IT4Innovations, Czech Republic

PRACE SoHPCContact
Karina Pešatová, Ostrava
Phone: +420 596 999 587
E-mail: karina.pesatova@vsb.cz

PRACE SoHPCSoftware applied
Unity3d, Blender

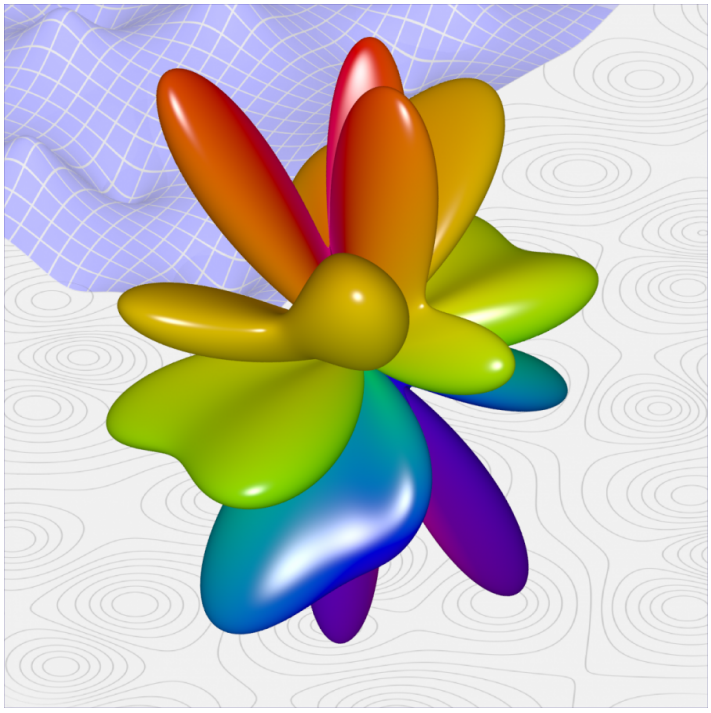
PRACE SoHPCMore Information
Unity3d Blender

PRACE SoHPCAcknowledgement
To Petr Strakoš and Milan Jaroš for the constant help and patience. Also thanks to IT4Innovations and their *free coffee* policy.

PRACE SoHPCProject ID
1617

# FMM for GPUs

*Johannes Pekkilä*



Want to know the latest tricks for simulating particle-particle interactions on GPUs? Look no further! In this article, we show how to accelerate N-body simulations with the fast multipole method using state-of-the-art GPUs.

The simulation of the movement and interaction of particles, from astronomical scales to atomic, is of great interest for scientists striving to unravel the mysteries of the universe. Since it is impractical to wait for billions of years for galaxies to collide, or follow the movement of individual atoms in a superheated plasma, we have to harness the power of massively parallel supercomputers to predict the likely outcomes. There is one problem though. Since each particle can interact with another in these types of simulations, the computational cost increases very quickly as we add more particles into a simulation.

Using a naive approach and a home computer, predicting the collision of galaxies would require time equivalent to the age of the universe, if not more to compute. Given this time, we might as well just wait a few billion years for the galaxies to collide by themselves!

Luckily, there is a way around this problem. Using the *fast multipole method* (FMM) and a highly parallel supercomputer, we can reduce the time needed for simulating the collision to just a few days. The bad news is, that figuring out how to solve the problem on state-of-the-art hardware in the most efficient way possible is far from trivial.

In this project, we seek to provide a proof of concept for solving the n-body problem on graphics processing units using the fast multipole method and a few extra tricks.

## Methods

The $O(np^3)$ approach to FMM consists of five steps. The bottleneck in these steps is called multipole-to-local operator, which in turn is composed of three subroutines. The subroutines consist of forward rotation, shift and backwards rotation. In this work, we focus on optimising the bottleneck - the rotation operation, on GPUs using CUDA. For more information on FMM, we refer the reader to [1, pp. 49–54].

For the rotation operation, we take a stack of triangles $\omega$, a single pyramid-shaped rotation matrix $D$ and a one-dimensional array of constants $e^{i\phi}$ as input. The output is a stack of triangles. These data structures are shown in Fig. 1.

$\omega$ and $e^{\phi}$ are complex numbers and the rotation matrix $d$ consists of real-valued tuples. We also need to introduce an operator we call *complex scale* $\circ$, which is defined as a complex number $(a, bi)$ and a real tuple $(c, d)$ as follows

$$(a, bi) \circ (c, d) = (ac, bdi) \ . \qquad (1)$$

Each element in the output can be solved in parallel using the following equation - with l and m signifying the index of an element within a triangle which consisting of $\frac{(p+1)(p+2)}{2}$ elements.

$$\omega'_{l,m} = \sum_{l=0}^{p} \sum_{m=0}^{l} \sum_{k=0}^{l} d_{l,m,k} \circ e_k^{i\phi} \omega_{l,m} \ . \qquad (2)$$

To be able to efficiently use GPU resources, this problem must be decomposed into sub-problems of a certain size. However, the optimal size is not obvious. We implemented four approaches to explore this, where the sub-problems consisted of computing the output for

- a whole triangle
- a whole column in a triangle
- a single element in a triangle
- several elements in a column of a triangle

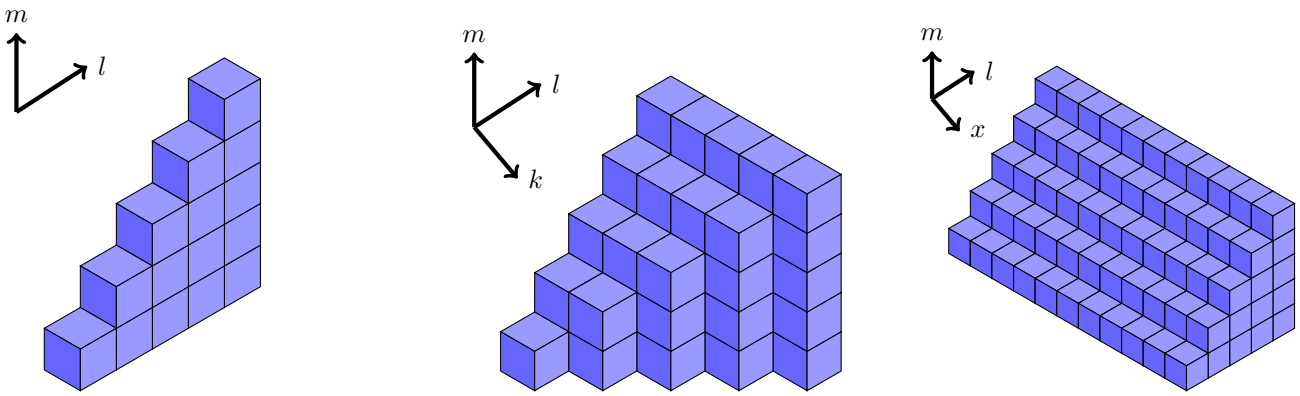



**Figure 1:** From left to right: A triangle ($\omega$ or $\omega'$), a rotation matrix ($d$) and several triangles stacked in the $x$-axis.

In our implementations we assumed that the same rotation matrix is used for computing all $\omega'$. In practice, there are 27 different rotation matrices. However, in our implementations the rotation matrix is reused within blocks of 128 threads, so we did not expect a severe performance penalty as long as the same rotation matrix is used to compute at least 128 output triangles.

## Results

The tests were run on the JUHYDRA cluster containing:

- 2× Intel Xeon E5-2600 @ CPUs
- 8× 8 GiB DDR3 @ 1600 MHz
- 2× NVIDIA Tesla K40m @ 745 MHz GPU
- 2× NVIDIA Tesla K20Xm @ 732 MHz GPU

The results were generated on a single Tesla K40 GPU with ECC enabled. We compared our solutions to the absolute minimum time we could theoretically achieve with the hardware. The absolute minimum was calculated by using a hypothetical machine, which has:

1. Infinitely large and infinitely fast caches
2. Free arithmetic operations
3. The maximum bandwidth of 268.22 GiB/s (the theoretical maximum of a K40m with ECC disabled)

The results are shown in Fig. 2 and 3. With sufficiently large number of triangles and $p = 10$, our best implementation of the forward rotation achieves 58% of the absolute maximum performance. It should be noted that the theoretical maximum bandwidth used in calculating the absolute maximum performance, is reduced approximately by 20% when ECC is turned on.[1] Figure 3 shows the comparison of the reduction in reads in the forward rotation, when either $\omega$ or $d$ is completely reused. For 4096 triangles and 27 different rotation matrices, it is beneficial to reuse the rotation matrix instead of $\omega$ when $p < 236$. This limit is further increased with the number of triangles.

## Discussion

We implemented and optimized the rotation operator for the FMM on GPUs. Our results show that the problem decomposition has a huge impact on performance when programming for GPUs. We showed that by prioritizing the reuse of the rotation matrix $d$ and solving multiple elements within a column is a very efficient way to solve the rotation operation on GPUs. With a sufficiently large number of triangles, the thread processors of the GPU have enough work to hide the latencies caused by arithmetic and memory fetches. As shown in Figure 2, we can reduce latencies in the kernel by having a thread solve multiple elements per column. Through unrolling, the threads can execute several independent instructions in a pipelined fashion, which has a similar latency hiding effect as increasing occupancy of the GPU.

Despite this, even with large unrolling factors the performance of our implementations suffer when the number of triangles is low. This occurs when there are not enough triangles to fill the GPU with work. With large unroll factors, we hit the limits of the caches and registers, which in turn causes data to spill to the main memory of the GPU. This degrades performance, since the main memory is slow compared to caches and registers.

As shown in Fig. 3, the same techniques used to accelerate the forward rotation, can also be used for shift and backward rotation. The rotation matrix is not used in the shift operation, so we prioritize the reuse of the input $\omega$ instead.

In future work, we will explore how the problem scales to multiple GPUs. Since the complete M2L operation is so close to the theoretical limit, there isn't much more we can gain by optimising it further.

## References

[1] Garcia, A. (2015). Parallel FMM on a GPU: a CUDA/C++ love story *[ONLINE]:* https://summerofhpc.prace-ri.eu/wp-content/uploads/2016/04/SoHPC2015_final_editorial.pdf

[1] NVIDIA. Cuda C: Best Practices Guide *[ONLINE]:* http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#theoretical-bandwidth-calculation

PRACE SoHPCProject Title

Shape up or ship out – You decide!

PRACE SoHPCSite

Jülich Supercomputing Centre, Germany

PRACE SoHPCAuthors

Johannes Pekkilä, [Aalto University School of Science] Finland

PRACE SoHPCMentor

Andreas Beckmann, JSC, Germany
Ivo Kabadshow, JSC, Germany

Johannes Pekkilä

PRACE SoHPCAcknowledgement

Great thanks to I. Kabadshow and A. Beckmann for continuous support in all matters, from tree structures to templates to coffee and snacks.

PRACE SoHPCProject ID

1618

## Appendix

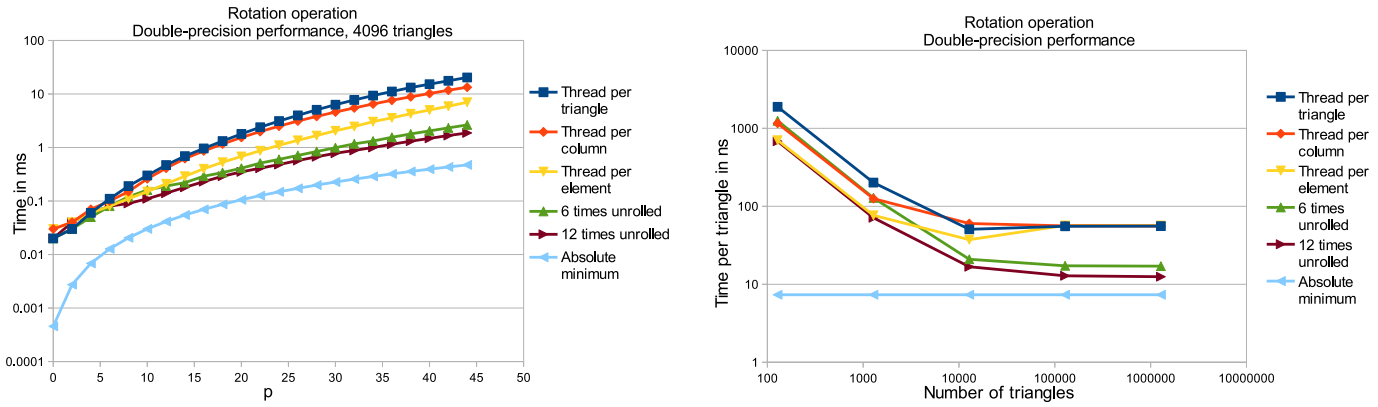



**Figure 2:** Performance results for the rotation operation using double-precision.

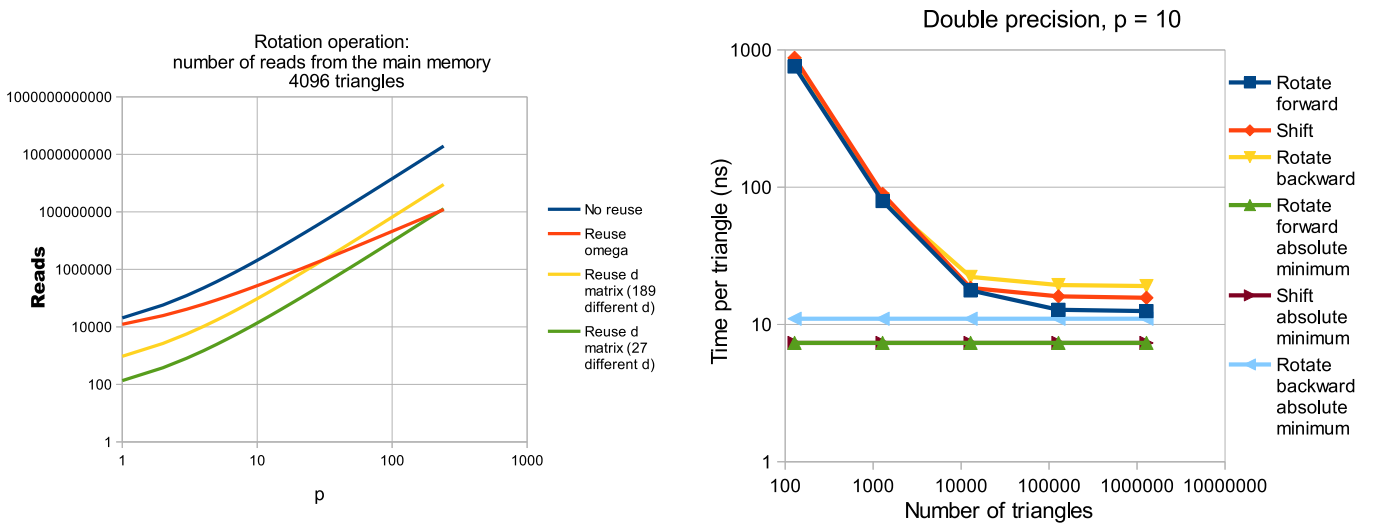
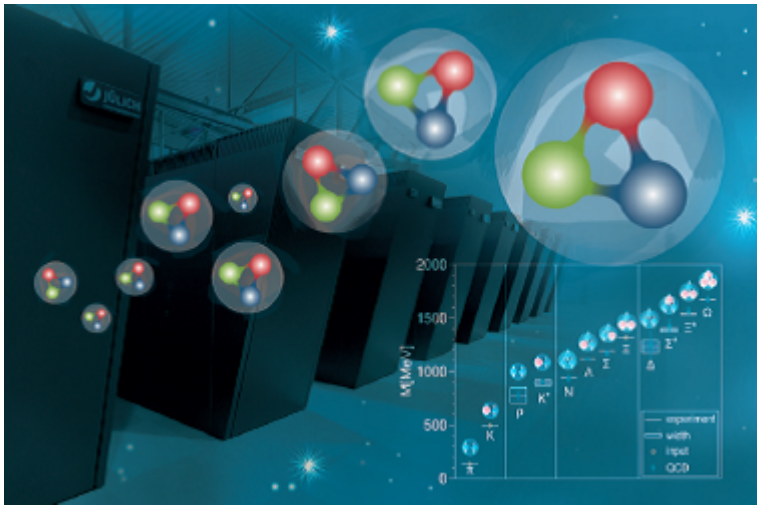



**Figure 3:** Left: Number of reads when reusing either $\omega$ or the rotation matrix $d$. Right: Results for the full M2L operation including forward and backward rotations and shift. Here a thread solves 12 elements in a column (along the $m$-axis) per thread in the rotation kernels. In the shift kernel, a thread solves 8 elements per row (along the $l$-axis).

Speeding up strong interactions of quarks and gluons on modern CUDA graphic cards and Intel Xeon Phi accelerators

# Phine quarks and cude gluons




***Peter Labus***

Lattice Quantum Chromodynamics is a discretized version of the theory of strong interactions, which in recent years has produced high-precision predictions for elementary particle and nuclear physics using large amounts of resources on world-class supercomputers. In this project we try to understand how to implement the most time-critical kernels of this theory on multi-core architectures such as graphic cards and accelerators.
To this end, we study a smaller toy model from the field of solid state physics which still has the full algorithmic complexity and depth as Quantum Chromodynamics. We identify a highly strong-scalable threading scheme and a large amount of vectorisation (SIMD) as the two main requirements for achieving high performance on an Intel® Xeon Phi™ coprocessor.

Lattice Quantumchromodynamics (LQCD) is the discretized version of the physical theory that is believed to described the so-called strong interactions — one of the four known elementary forces in the universe. It describes the movement and interaction (the *dynamics*) of elementary particles called quarks. Their charge is called colour (Greek *chromos*) and determines how they combine into heavier particles - such as protons and neutrons, which then themselves form the nucleus of the atom. The particles that are responsible for this sort of binding mechanism are called gluons because they stick or *glue* the quarks together. Interestingly enough, these particles are also charged and interact with the quarks and amongst each other also. As a result, the equations which describe their dynamics are non-linear and the theory cannot be solved easily with analytic methods. However, discretizing the underlying continuous space-time structure and looking only onto a finite subset of points provides an excellent framework to study the theory on the computer. The discrete version of space-time assumes the form of a grid or *lattice* — which explains the origin of the name of the theory.

Mathematically, QCD is fully described by an integral (the so-called path integral or partition function) which can be manipulated to extract information from the theory and calculate observable quantities which could then be compared with data from experiments. Numerical methods to estimate integrals have a long history. Standard methods turn out to be computationally expensive when the dimensionality of the domain of integration is big. For these kind of integrals, a sophisticated but simple technique which is based on pure randomness turns out to be most efficient. This is known as Monte Carlo Integration — with reference to the infamous casinos of the Principality of Monaco. In essence, one lets the computer roll a (very clever) dice many times to select the portions of relevance of the integral and then sums them up. This can be shown to be a sufficiently fast converging estimate of the full integral.

In LQCD, the preferred variant of this method is called *Hybrid* or *Hamiltonian Monte Carlo* (HMC) algorithm. The former name indicates that the algorithm merges two of the three most influential ideas in Scientific Computing of the 20th century — the *Metropolis Monte Carlo* algorithm and *Molecular Dynamics* (the third being the *Fast Fourier Transform*). The latter name however is somewhat more suggestive since it refers to the specific system of equations that is used for the Molecular Dynamics (MD) Integration. MD is the procedure of integrating a system of differential equations (i.e. *equations of motion*) by means of discretizing the differential operators, which transforms the equations into simple algebraic ones.

To put it in a nutshell, to solve the theory one has to calculate an integral — which, using a computer can be done by integrating the relevant equations of motion (eom) and then using the HMC algorithm. The eom describe how the quarks move or *propagate* in space-time under the influence of the gluons. It turns out that this is described by the inverse of an intricate differential operator (the Dirac operator or *Dslash*) in the continuum theory, which however becomes a (very sparse) matrix in the discretized version. This is why one has to solve a lot

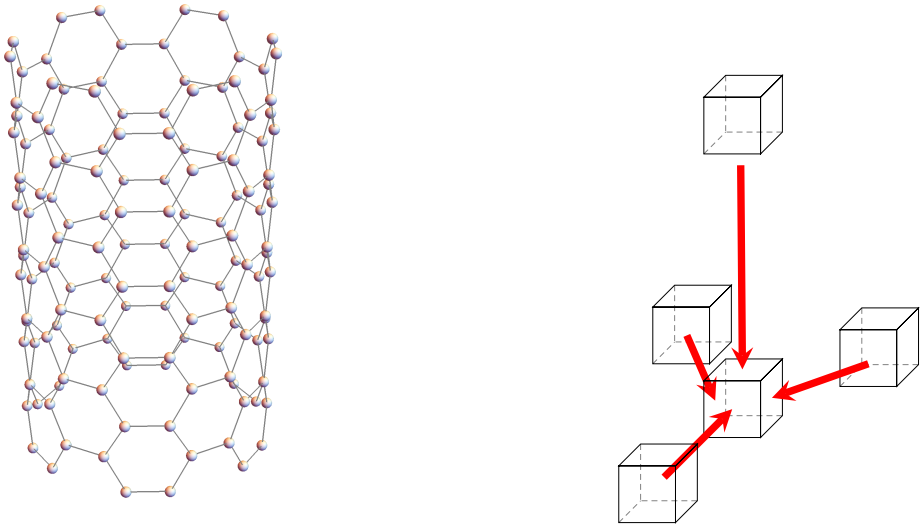


**Figure 1:** Left: One possible geometry of the lattice: a honeycomb layer rolled up to a tube. Right: The stencil operator.

of sparse systems of linear equations involving this matrix.

The best way to do this is to use so-called iterative solvers, which solve the system of linear equations only up to a finite precision, but have the great advantage that the matrix involved is not changed during the process. This allows for the possibility of implementing the matrix multiplication "by hand" and restricting the floating point operations to only those that involve non-zero elements.

In LQCD matrix multiplication, this is simply what computer scientists call a *stencil operation* — that is, for each and every point of the grid the operation is a combination of multiplications and additions involving only a few adjacent elements of the point in question (cf. Figure 1 as well). When it comes to LQCD, one may simply count the number of floating point operations (flops) involved, as well as the number of data elements one has to read or write from/to the main memory or cache. The resulting ratio turns out to be very small in terms of flops per bytes of moved data. Since the gap between the velocity of calculation and movement of data continues to increase, the memory bandwidth of a computational system will be a limiting factor. This is one of the great challenges in Lattice QCD.

Lattice QCD is one of the most computationally expensive fields in scientific computing and uses large fractions of the available supercomputer resources worldwide. Walltimes for the generation of Monte Carlo configurations can easily reach a year on most supercomputers. In particular, several new important results in hadron physics are expected to have demands of hundreds of Teraflop-years or even Petaflop-years.

It is therefore very desirable to have simpler and smaller toy models to test the impact of algorithmic variations and hardware specific considerations in much smaller time scales. One of the areas where one may find such examples is the field of solid state physics, where many phenomena are naturally described using grids - due to the rigid structure of the materials one wants to study. Furthermore, interactions are usually local, i.e. appropriately modelled by nearest neighbours, which result in very similar stencil operators for the Dslash matrix multiplications.

In this project, we are interested in optimising an existing code base that simulates nearest neighbour interactions of electrons, supplemented with a screened Coulomb potential. The electrons are spatially confined to honeycomb structures, similar to two-dimensional planar carbon layers of graphene.

The goal of the project was to optimise this program for modern many-core and multi-core architectures which come with big vector registers - such as the Intel® Xeon Phi™ coprocessors. To use their large registers, time-critical parts of the code have to "vectorise", i.e. *single instructions with multiple data* (SIMD) have to be used. This means that the program has to be parallelised, even to the level of a single computing core. To use a large number of available cores, one has to make sure that all cores execute similar amounts of work in parallel.

## Methods

As with most matured code bases, we had to invest a fair amount of time to study the serial (unthreaded) program and to get a principal understanding of function hierarchy and call structures. Profiling tools was used to identify hot spots and bottlenecks of the code. A great number of improvements were made by removing constant expressions out of for-loops, minimizing branching, reducing the number of re-allocations of large arrays, adding compiler optimization among other techniques. After implementing these, code profiling revealed that the predominant part of the program execution was indeed spent – as expected – in the matrix multiplication kernel.

Originally, the matrix was allocated in its entirety in the code and then parsed to a sparse layout. Since the multiplication can however be described as a five-point stencil operation (cf. Figure 1) — similar to the case of LQCD — the matrix need not be allocated at all. The multiplication can rather be described as a sum of shift operations and multiplications with constants. One pitfall of not allocating the matrix is however the following. The nearest neighbour elements involved in the matrix multiplication are not know at compile time. They strongly depend on the geometry used, which in turn should be freely adjustable in a configuration file that is passed to the executable. To resolve this issue, we originally decided to implement a python script that generates C++ code with pre-calculated array indices for all relevant cases. This however turned out to be rather cumbersome and hard to maintain. Therefore, we implemented a lookup table from which the positions of the nearest neighbours could be read out at execution time. This did not cause any measurable overhead — possibly because the number of instructions was greatly reduced at the same time.

Implementing the matrix multiplication as a stencil operator presented a large improvement in terms of performance and in particular in terms of the memory footprint. In this way, we could reduce the required memory by several orders of magnitude (cf. Figure 2) and were even able to fit the entire iter-

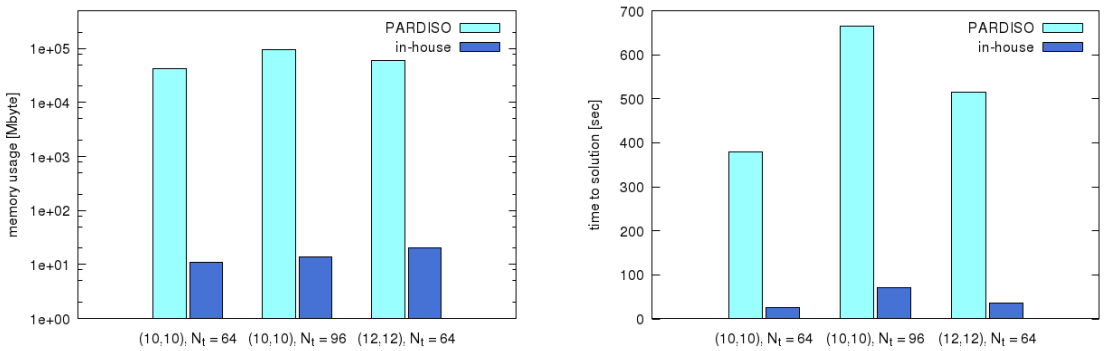



**Figure 2:** Left: Memory usage for version with external library (Pardiso) and without. Right: Time to solution for the same geometries.

ative solver into the L3 cache.

After restructuring the serial version in this way, the code was ready to be parallelised. To be able to use an entire CPU node or an entire Intel® Xeon Phi™ coprocessor it was necessary to "thread" the program - that is to use a shared memory parallelisation scheme. We initially decided to use OpenMP, because it is a simple and fast to implement Pragma-based C++ language extension. The final version showed good strong scaling up to half of a two-socket 24 core Xeon™ node.

Equipped with a fully parallel version of the program, the expected memory footprint, bandwidth and throughput which had been calculated could be compared with data from detailed profiling. Of particular interest was the memory bandwidth which — as explained above, is often the limiting factor for stencil based codes. Comparing the peak performance of the available hardware with the measured throughput can give an indication if vectorisation (SIMD) can benefit the code, or if the memory bandwidth is already saturated within the (core) serial version.

The OpenMP version of our code showed that the memory bandwidth was not saturated at this point. Since the Xeon Phi™ coprocessor has large 512-bit vector registers, one expects that successful SIMDization can massively increase the throughput.

The implementation of this feature however was a non-trivial task. This because the stencil operator in question is fairly irregular, depends strongly on the geometry and boundary conditions and differs from site to site. This is why the direction of vectorisation had to be considered carefully, in particular because it was not clear if the rather small problem sizes of interest could benefit from the implementation of OpenMP threading and SIMD in the same direction.

Since we did not want to change the main memory layout of the data, mainly so we wouldn't need to change major fractions of the existing code, there was no way to avoid implementing the threading in the time direction. We additionally decided to use a common and easily implementable vectorisation procedure which goes under the name of *site fusion*. In this approach, vectors are formed by cutting the outer most dimension (time in our case) into pieces of equal length and 'laying' them above each other. The number of pieces correspond to the number of elements that fit into the register, such that the combined $n^{th}$ elements of all pieces form the $n^{th}$ vector. Special care had to be taken for the boundary elements which had to be shuffled in a clockwise or counter-clockwise manner using vector intrinsics. Using threads and vectors in the same direction did however have the disadvantage that small problem sizes could not benefit from the additional throughput, because the thread synchronisation became the limiting factor.

Furthermore, we discovered that we could use a simpler linear solver — a conjugate gradient (CG) solver, instead of the original bi-conjugate gradient (BiCG) solver. The simpler CG solver assumes additional properties of the matrix (which turn out to be satisfied) but uses only half of the memory and converges at roughly half of the iterations. We also implemented the CG solver in single precision in order to use it as a pre-conditioner in a mixed precision solver that can achieve further speed-up through an increase in the memory bandwidth. We could however not observe any impact. One possible explanation for this is that one might be limited by either cache misses or instructions already.

## Conclusions and Outlook

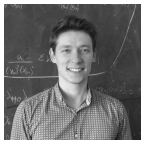
Originally, the code shipped in two versions - one with customised functions and another one which used a library for sparse matrices (Pardiso) to solve the linear equations. With our final *in-house* version we managed to obtain both a speed-up of a factor 10x to 14x, as well as a reduction of the memory footprint by 3 to 4 orders of magnitude (cf. Figure 2). The improved memory demands further allow us to simulate much bigger problem sizes than before. One issue that we want to deal with in the future is the improvement of the cache use. The re-use of data in the cache, which is essential for memory bound problems, may be improved by keeping elements that are adjacent in the grid closer together in the cache. This may for instance be realised by using so-called space-filling curves.

### PRACE SoHPCProject Title

Phine quarks and cude gluons

### PRACE SoHPCSite

Jülich Supercomputing Centre, Germany



Peter Labus

### PRACE SoHPCAuthors

Peter Labus, SISSA Trieste, Italy

### PRACE SoHPCMentor

Dr. Stephan Krieg, JSC, Germany

### PRACE SoHPCContact

Labus, Peter, SISSA Trieste
E-mail: Peter.Labus@sissa.it

### PRACE SoHPCSoftware applied

GNU/Linux, GCC, Perf, Valgrind, Vim

### PRACE SoHPCAcknowledgement

I would like to thank S. Krieg, T. Luu and E. Gregory for insightful discussions. I am also grateful for the hospitality of the JSC during the duration of this project.

### PRACE SoHPCProject ID

1619

### PRACE SoHPCReferences

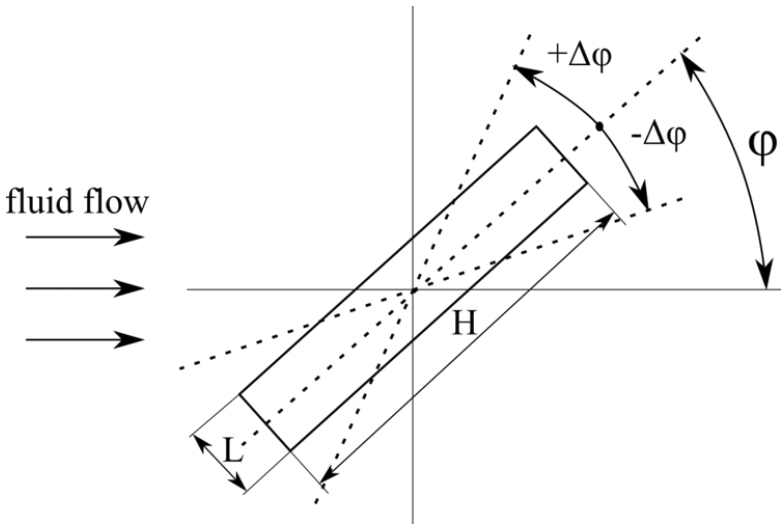
[1] T. Luu and T. A. Lähde, "Quantum Monte Carlo Calculations for Carbon Nanotubes," Phys. Rev. B **93** (2016) no.15, 155106

[2] D. Smith and L. von Smekal, "Monte-Carlo simulation of the tight-binding model of graphene with partially screened Coulomb interactions," Phys. Rev. B **89** (2014) no.19, 195429

[3] R. Brower, C. Rebbi and D. Schaich, "Hybrid Monte Carlo simulation on the graphene hexagonal lattice," PoS LATTICE **2011** (2011) 056

# The Easy guide to CFD

*Sam Hewitt*




Computational Fluid Dynamics involves a lengthy process starting with modelling a geometry, building a domain, creating a mesh, simulating the flow before finally analysing the results. This lengthy process can be automated using two open source software.

Like many problems in engineering, fluids are described by a series of equations which are known as the Navier-Stokes equations. These are made up of equation 1 and 2 - where $\rho$ is the fluid density, $P$ the pressure, $t$ the time and $U$ the velocity vector $uvw^t$.

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho U) \quad (1)$$

$$\frac{\partial U}{\partial t} + \nabla.(\rho UU) - \nabla\tau + \nabla P = \rho b \quad (2)$$

These equations tell us that at any two points in time in any bound system, the system must have the same mass and the same momentum.

For very simple problems these equations can be solved analytically, but for more complex problems they need to be solved numerically. To do this, we separate the spacial and time domain into discrete areas/volumes and steps. In moving from a continuous domain to a discrete one, we are introducing the possibility of numerical errors, so care must be taken. The two equations above can then be altered to provide us with a set of linear equations which can be numerically solved over the system for each time step. This is where supercomputers come in. As problems get more interesting and more complex, large numbers of small cells are required to maintain a good level of accuracy. In doing so, systems of equations grow to sizes that make it unfeasible to calculate a result on a single desktop machine.

## Problem description

In Slovenia and Croatia, roads and bridges are often subjected to high speed cross flows that can impart large drag forces on passing vehicles - making it dangerous and difficult to drive. One of the proposed solutions to this problem is the use of wind barriers which have been studied in a range of manners. This project considers how the use of porous wind barriers can manipulate the flow and reduce the dangerous forces imparted on vehicles. An example of one of these barriers can be see in Figure 3. It can be seen that a series of bars are used to distort the incoming flow.

## Software

This project involved the use of three open source software, **OpenCASCADE**, **OpenFOAM** and **ParaView**. These three C++ based programs are easily accessible and can be run from the kernel. Before describing the software in more detail, we present the basic work flow of a CFD problem - which will make it easier to visualise how these pieces of software are used. Figure 1 shows the three main steps in any CFD Problem.

### OpenCASCADE

This is a collection of C++ libraries that are used for building and manipulating geometric models. It has a python wrapper that makes it easier to use for users who are not used to C++. This is the pre-processing step (red) from Figure 1. However it will not be used for

the meshing step in this problem.

### OpenFOAM

This is a C++ library that can be used for the "Mesh the domain" step in the pre-processing section (red) and analysis step (blue) from Figure 1.

### ParaView

This software is used by many applications to read data and visualise results. It accepts .foam formatted files that contain all the information from an analysis step. It will be used for the post-processing step (green).

## Solution

This section describes methods which are used to make a CFD engineers life easier.

**Build the Geometry** - The geometry is produced by OpenCASCADE using the shape and file writing libraries. These libraries allow easy manipulation of geometric values and allow the object to be saved in STL format. To change the geometry of the barrier, the user need only change one value and a range of stl files are created for the new wind barriers. For the problem in this report the angle of the bar was chosen as the critical variable. Therefore four angles were used as input to the python script and 4 stl files were the output. Figure 3 shows a range of bar angles at a 2D cut and a full barrier.

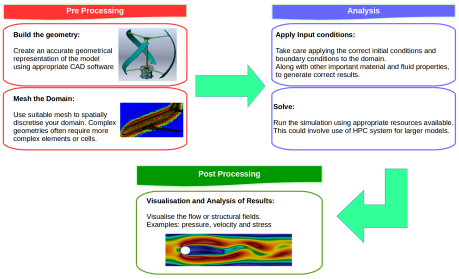



**Figure 1:** Pre-processing, Analysis and Post-processing steps in CFD problems.

## Mesh the Domain

The output of the python script is four stl files that are copied to the correct directories. These are the directories in which OpenFOAM can find the stl files. In order to build a problem that OpenFOAM can solve, a domain must be generated and a mesh produced. To build the domain, the user uses another python script that takes the height dimension of the barrier as its input and it outputs a file(blockMeshDict) that can be read by OpenFOAM, to define the domain. This is used in combination with a dictionary - called snappyHexMeshDict, which describes the overall meshing constraints to mesh the domain using hexahedral elements. This step is done in parallel by decomposing the problem into a set of user defined volumes. An example of a final mesh is shown in Figure 2.

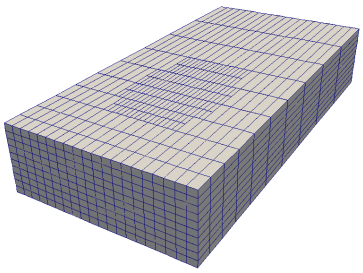


**Figure 2:** Example Mesh

## Analysis

The analysis step is the easy part. At this stage in the process, the user has created a geometry and defined the domain using a python script and 6 input variables. They have then selected how many cores they want to use and used the OpenFOAM meshing algorithm to automatically mesh the domain in parallel. All that remains is to select which OpenFOAM solver to use, and type run. In reality, the run script does a few extra things but a user doesn't need to to worry about these, as someone else has already written the script.

## Visualisation

OpenFOAM, like OpenCASCADE has a series of python utilities that can be used to output visual results. One very useful tool is **pyFoamPVSnapShot**. Having created four different angled wind barriers the results of these can be compared. After importing one of the results files to Paraview and configuring the options, the chosen view can be found. By saving the state and calling the pyFoamPVSnapShot python script using the saved state and the results files, four images of pressure and velocity at the latest time step are produced. Figure 3 shows the results from the test cases ran by the author.

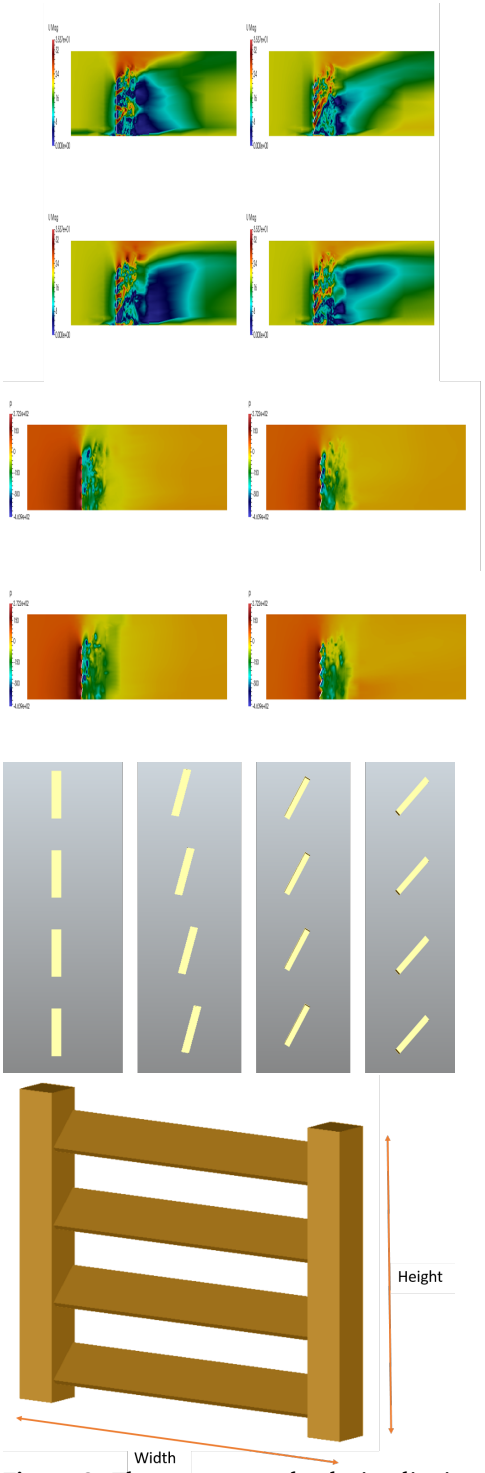



**Figure 3:** The pressure and velocity distributions from a range of bar angles, along with a full geometric model of the barrier and a 2D slice of the bar in the x-y plane

## Discussion & Conclusion

In conclusion, I have developed a versatile and flexible method to build a geometric model in OpenCASCADE and subsequently offered an easy way to mesh and simulate this model using the capabilities of OpenFOAM and some added python scripts. To finish, I have made use of Paraview capabilities to compare a series of results for similar simulations.

## Acknowledgements

The author would like to thank PRACE for the opportunity and funding for this project, along with the site coordinator and project supervisor, Leon Kos and Marijo Talenta, who were incredibly kind and helpful during the project.

## References

[1] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.

PRACE SoHPCProject Title

The CFD devil is in CAD details

PRACE SoHPCSite

University of Ljubljana, Slovenia

PRACE SoHPCAuthors

Sam Hewitt, [University of Manchester] England

PRACE SoHPCMentor

Marijo Talenta, LECAD lab, Slovenia

Sam Hewitt

PRACE SoHPCContact

Sam, Hewitt, University of Manchester
Phone: +44 7949914899 E-mail:
sam.hewitt@postgrad.manchester.ac.uk

PRACE SoHPCSoftware applied

OpenFOAM,OpenCASCAE and Paraview

PRACE SoHPCMore Information

OpenFOAM CFD

PRACE SoHPCAcknowledgement

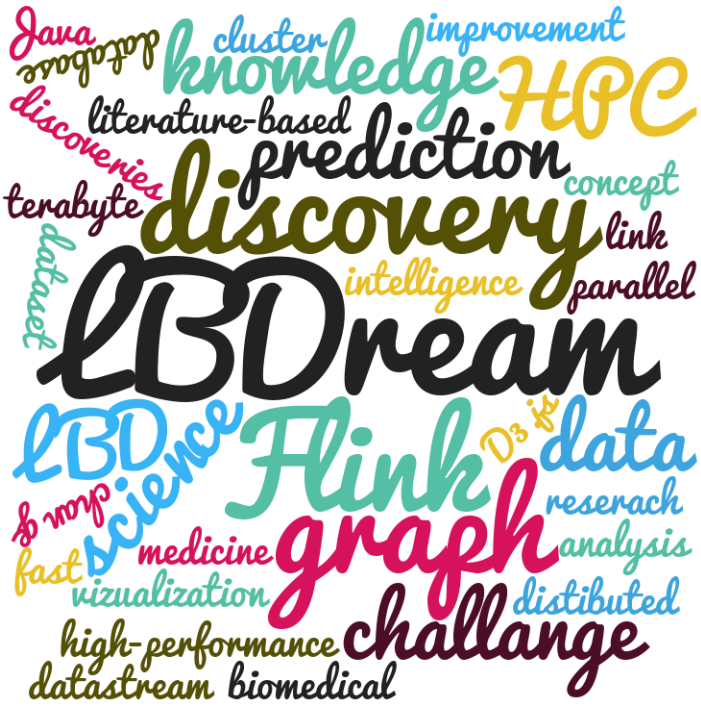
The author would like to thank the project supervisors Mario Talenta and Leon Kos for the help provided and PRACE for the oppurtunity to undertake such a fantastic project.

PRACE SoHPCProject ID

1620

Link prediction in large-scale literature-based discovery networks with Apache Flink framework

# Literature-based discovery at scale!




***Mateusz Lango***

Big Data can change the way we identify scientific discoveries. Big data analytics upon huge volumes of data are probably sufficient to make significant discoveries otherwise unnoticed due to a lack of proper analytical tools. In this project, I created an application which uses artificial intelligence techniques to explore a large database of scientific medical publications looking for discoveries missed by humans.

We live in the era of big data. This statement is nowadays heard more and more often in the media. But from where does all this data comes from? Let me give you an example. Ten years, ago almost everyone was paying for their shopping using cash, whereas nowadays people usually pay using their credit or debit card. This may seem like a small change in our lives, but it causes the production of big amounts of data! Every card transaction is stored in the databases of banks but also on the servers of card providers and in the interbank transfer exchange systems. What's more, all these systems produce log files and other data artifacts. This is a lot of data, especially taking into account that in some European countries the number of card transactions grows exponentially in time.

Recently, the widespread use of mobile devices has changed the way we live and caused the generation of even bigger amounts of data. Using data from our mobile phones, companies are able to estimate the traffic on public roads and recommend us a faster route. They can also warn us about the delay of a bus or propose good restaurants nearby.

Big data has changed the way we live, but has it changed the way we are doing science? Not yet.

## The Big Data challenge in Science

At first glance, one can think that in science we are using the most modern technologies, so big data should not be a problem for us. Indeed, in science we do not have problems with generating huge amounts of data. As an example, the Large Synoptic Survey Telescope which is currently under construction in Chile will create high-resolution photos of the sky. The quality and size of these photos will be so high that 30 TB of disc space will be required to store such an image. How much data does this amount to?, you may ask. Well, the storage space required for the human genome of 30 thousand people or for storing 60 million books is of similar size. Do you think this is a lot? In 2020 it is planned to launch the Square Kilometre Array which will generate 1,048,576 TB of data every second.

However, it is impossible for a scientist to analyse such a huge amount of data by hand and techniques for automatic analysis of scientific data are still lacking. Because of this, there is an increasing interest in automatic knowledge discovery techniques using machine learning, computational and artificial intelligence, decision support systems and others.

## Literature-based Discovery

One of these ideas is the concept of Literature-based Discovery (LBD). In LBD we assume that a scientist is not aware of every single paper which appeared in their discipline and the goal is to help them by analysing these automatically.

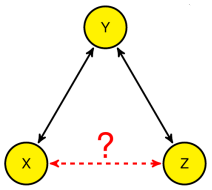



**Figure 1:** The problem of link prediction

I explain how LBD works through an example. Let's assume that some researchers discovered that medicine A cures the disease B. Later, another scientist has written a paper where he claims that disease C is somehow related to disease B. However, no one is aware of the possible interaction between medicine A and disease C. Maybe even by applying this drug we can cure this disease? Such a discovery is just waiting to be found - no stroke of genius is required at all!

## How to find a discovery in data?

In my project, we are concentrated on exploring the MEDLINE database of medical scientific papers. MEDLINE is the largest bibliographical database in the world and it's freely available.

Each MEDLINE entry is annotated by hand with around 12 Medical Subject Headings (MeSH) terms which aim to describe the content of the article in a concise way. The examples of MesSH terms are "Postmenopause", "Cardiovascular Diseases/chemically induced" or "Estrogen Replacement Therapy/adverse effects".

Using MeSH terms we can build a knowledge graph in the following way. We first add a node to the graph for every MeSH term in the database. We then construct the edge between nodes (terms) if the terms co-occur in at least one scientific paper. This way we have a graph of terms, but naturally, the majority of the terms are not connected to each other. If we could predict which nodes in the graph should be connected in the future, then actually we would be able to automatically make discoveries! Note, that - just like in our example, two vertices can represent the cure and the disease and the edge is the possible new publication which claims that they are related! Hence, we used linkage prediction techniques to automatically find promising new connections between MeSH terms.

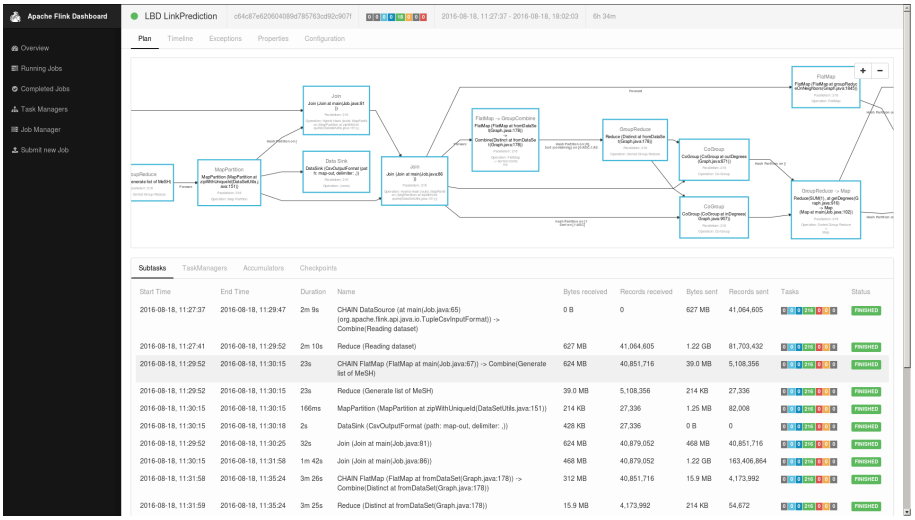



**Figure 2:** The dashboard of Apache Flink with visualised data-flow of our linkage prediction algorithm.

## The need for HPC

In order to decide, which connections in the graph are probable and which are not, the computer has to analyse all of them. Unfortunately, the number of possible connection is very big. For instance, if our graph contains 200 000 nodes, then there are almost 20 000 000 000 possible edges!

This makes such calculations computationally demanding as it requires great computational power, which can be provided by high-performance supercomputers.

One of the challenges of the project was to implement linkage prediction in a distributed way, i.e. to write a program which can use multiple computing nodes of a supercomputer. To achieve this I decided to implement the project in one of the big data frameworks which implement the MapReduce paradigm.

MapReduce consists of three steps: Map, Shuffle (which is done automatically) and Reduce. During a map stage, each line/object of the input file is transformed into one or many key-value pairs. For a particular line/object the map operation does not depend on other lines/objects, hence it can be executed at any time and at any node. Later, during the shuffle, each node sends all previously produced key-value pairs to other nodes. However, they are transported in a way which ensures that all pairs with the same key will be transported to the same node - thus making further processing much simpler. Finally, the reduce stage reduces the group of key value pairs with the same key to one value.

This sounds pretty complicated and specific, but MapReduce allows us to write distributed applications in a relatively simple way and it has come to be a standard in the implementation of big data processing applications.

I looked into three MapReduce-based frameworks for the implementation of my system - Hadoop MapReduce, Apache Spark and Apache Flink. After reviewing documentation and performing some initial performance tests I decided to use Apache Flink.

## Implementation in Apache Flink

Since there are millions of possible connections in a graph I implemented the calculation using the Apache Flink framework. Flink is an open source big data processing engine which treats data as a data stream. I was working with standard, static databases, but even then Flink considers it as a data stream of a constant and finite size - which allows Flink to perform more optimisations than standard big data frameworks. In particular, in many cases Flink does not need to wait for the termination of the previous stage of calculations to start the next one. Since the previous stage output is a data stream, then just after the first result is produced it can be further processed without waiting for the rest of the results. Another advantage of using Flink is that

it comes with a library for graph processing called Gelly. Gelly supports transformations and mutations on graphs, as well as more advanced algorithms

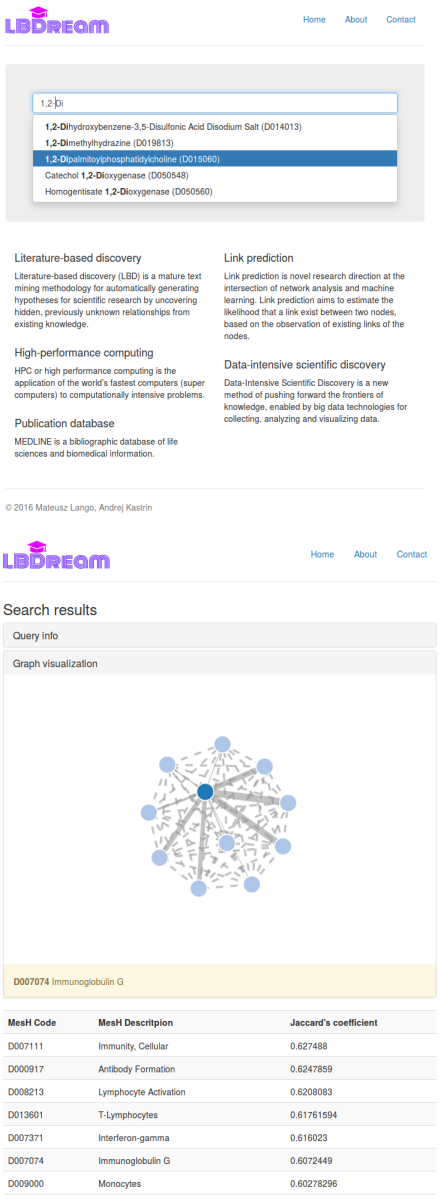



**Figure 3:** The main page (top) and view of search results (bottom) of LBDream.

such as PageRank, Triangle Count, Label Propagation, amongst others.

Flink provides programmatic APIs for several popular programming languages such as Java, Scala, and Python. In the project, the Java API was used, which included modern features of Java 8 such as lambdas and diamonds. I also used Flink-specific function annotations to allow Flink's engine to better optimise my code during the construction of the execution plan.

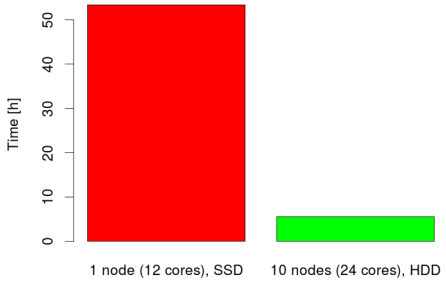



**Figure 4:** The execution time of linkage prediction algorithm on one and ten nodes.

We experimented with Flink using one PC (of 12 cores, 64 GB RAM, SSD) and on a cluster of 10 nodes (each with 24 cores, 48 GB RAM, HDD) with Flink set to standalone mode - which sets the number of task slots to be equal to the number of available cores. The comparison of execution times on one node and on the cluster is presented on figure 4 with the results showing that Apache Flink scales very well.

## LBDream

After processing the whole graph in Apache Flink, the resulting file is imported to a specially developed Web application called LBDream. LBDream is written in a python-based Django web framework and its main functionality is to search and explore linkage prediction results. Besides python, I also used Twitter Bootstrap, D3.js as front-end libraries and PostgreSQL as a database engine.

On the main page of LBDream (see figure 3), the user can search for discoveries by typing MeSH code or MeSH name in the input field (which also has auto-completion capabilities). The system enables search using four linkage prediction methods - Jaccard similarity, Adamic/Adar, Preferential Attachment and Common Neighbours.

The result of the search is then presented both as a table and as an interactive graph visualisation. On a special tab, the user can see specific information about the search method and selected MeSH terms (position in the hierarchy, specificity etc.).

I hope to make LBDream accessible to all medical researchers on the Internet soon.

### PRACE SoHPCProject Title

Link prediction in large-scale networks with Hadoop framework

### PRACE SoHPCSite

University of Ljubljana, Slovenia

### PRACE SoHPCAuthors

Mateusz Lango, Poznan University of Technology, Poland

### PRACE SoHPCMentor

Andrej Kastrin, Faculty of Information Studies in Novo mesto, Slovenia


Mateusz Lango

### PRACE SoHPCContact

Mateusz Lango, Institute of Computing Science, Poznan University of Technology, ul. Piotrowo 2,
60-965 Poznan, Poland
Phone: +48 61 665 23 33
E-mail: mateusz.lango@cs.put.poznan.pl

### PRACE SoHPCSoftware applied

Apache Flink (HPC processing), Django, PostgreSQL, D3.js, Twitter Bootstrap (webapp)

### PRACE SoHPCMore Information

flink.apache.org

### PRACE SoHPCAcknowledgement

I would like to thank to my site coordinator, Dr. Leon Kos from University of Ljubljana for his help and support.

### PRACE SoHPCProject ID

1621


## About PRACE

The Partnership for Advanced Computing in Europe (PRACE) is an international non-profit association with its seat in Brussels. The PRACE Research Infrastructure provides a persistent world-class high performance computing service for scientists and researchers from academia and industry in Europe. The computer systems and their operations accessible through PRACE are provided by 5 PRACE members (BSC representing Spain, CINECA representing Italy, CSCS representing Switzerland, GCS representing Germany and GENCI representing France). The Implementation Phase of PRACE receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement RI-312763 and from the EU's Horizon 2020 research and innovation programme (2014-2020) under grant agreements 653838 and 730913. For more information, see www.prace-ri.eu.


www.summerofhpc.prace-ri.eu