PRACE

Partnership for Advanced Computing in Europe

Summer of HPC 2015

www.summerofhpc.prace-ri.eu

A long hot summer is time for a break, right? Not necessarily! PRACE Summer of HPC 2015 final reports by participants are here.

HPC is hard?

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.

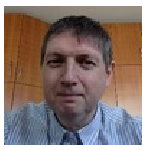
With help from all the mentors and the PRACE support staff, we have spent a wonderful summer working on various supercomputing projects. What a wonderful experience Summer of HPC 2015 has been! Since the sun is shining brighter and gently warms our hearts, there seem to be more smiles on faces everywhere. This is definitely true for all the Summer of HPC participants that have left their HPC sites. There were no vacations for them over the summer. The proof being the amazing projects they have created. I must thank them for all the wonderful moments I have spent with them, both teleconferencing and during site coordination.

I was involved in the PRACE *Summer of HPC* programme more than I should have been, my wife said. But then, I should explain why I was so excited and enthusiastic about Summer of HPC. It's because I had a similar experience when I was a student and participated in an exchange program with John Hopkins University. I've remembered that time as one of the most pleasant experiences where I was enlightened about many things. PRACE *Summer of HPC* made it possible for me to revive those times again and again. That's why I am singing PRACE Summer of HPC will go on and on! And I believe I am not alone as many were deeply touched by the wonderful idea of Summer of HPC. The articles that the students have written will prove that. It will demonstrate that the programme we ran can be as effective as all the science that is coming out of supercomputers by eminent scientists. Because it broadens the overall understanding on Who is capable of governing supercomputers? That's why I suggested putting student photos next to each project colophon. I will print complimentary copies of this publications and challenge people who say “HPC is hard!”. Yes, it is, but look at what can be accomplished in just two short months. *Summer of HPC* is thus not just an outreach program, as some might think. It is a science for reality. Training activity *in persona*. Students and mentors put all their knowledge and time into projects . And some results will prove to be quite useful for a broader HPC community.

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

Contents

PRACE SoHPC2015 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

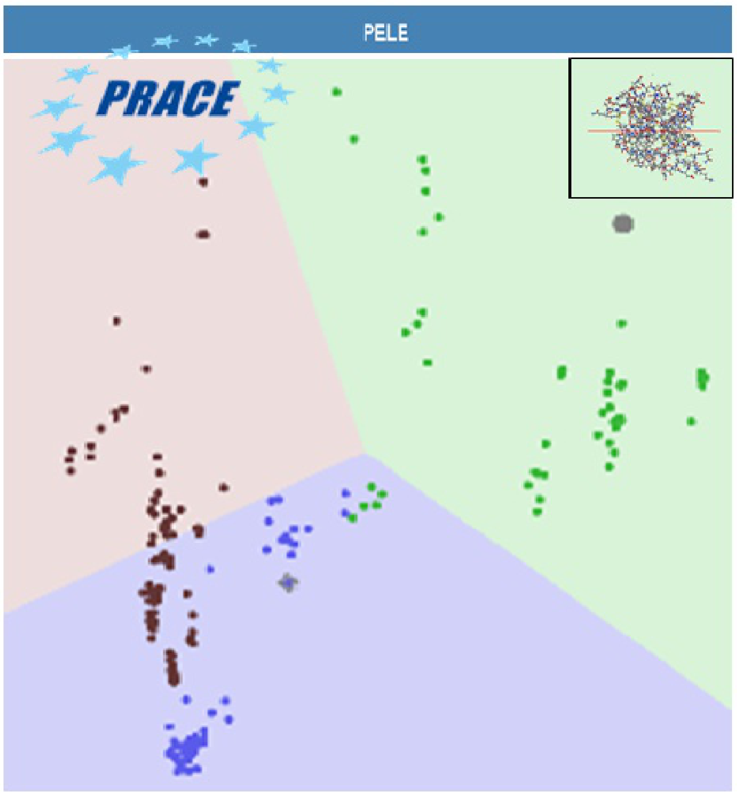

Leon Kos

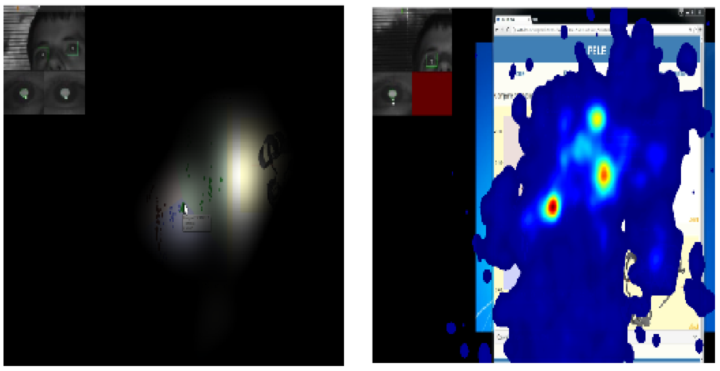
Developing an infrastructure for real time visualization, analysis, and steering of HPC molecular dynamics simulations

GUI for PELE

Simos Kazantzidis

Every day, the need for high-throughput visualisation and analysis is growing rapidly. That's why scientists who work on docking problems need an efficient molecular tool for high-throughput screening. With the assistance of PELE, a simulation software which provides the data pre-processing and analysis, we will visualise PELE's clustering results in a user friendly graphical interface, in order to evaluate the best matches.





Molecular docking is a well known procedure that calculates and predicts "molecule-molecule" combinations such as the ligand-protein. PELE's simulation software gives us the opportunity to model such an interaction. Using our tornado server and designing a user-friendly graphical User Interface using WebSocket communication technology, in order to achieve a client-server interaction, we are able to successfully visualise, cluster, and evaluate PELE's results.

Introduction

In the field of Bio-informatics and, more specifically, the dynamics of molecules, the docking procedure is used in order to predict a molecule's orientation so that it is able to bound to other molecules. Scoring functions can be calculated to show the strength of such a binding. Molecular docking refers to molecules such as proteins, lipids, nucleic acids, and others.

The most common docking procedure involves the goal of the ligand-protein combination in the area of drug-design. PELE, Protein Energy Landscape Exploration, is a well known simulation software which provides such modelling by using protein structure prediction algorithms and Monte Carlo sampling. By inserting program database files (pdb) with the protein structures of interest, the PELE program starts analysing and computing the best fitting sites and presents an atomic view of this interactions.

Our work here consists of developing a graphical interface which can visualise and cluster PELE's results in order to evaluate docking solutions or to better understand the binding process. The user has the ability to determine the clustering parameters. We do save his preferences in a message format known as json message and forward it to the tornado server by using Web Sockets. The json message is then delivered to the PyProCT program where the clustering process can be accomplished. The results follow exactly the reverse path and finally can be successfully visualised.

Methods

A combination of D3.js library, Web Socket client/server web technology, and JSmol visualisation technique is used, in order to obtain our project's goals. More specifically, D3.js is a JavaScript library that helped us create and build the required data visualisation framework by including useful JavaScript functions and utilities and giving us the opportunity to work with existing web technologies like HTML and CSS. Web Sockets technology allowed the creation of a TCP socket connection between the client(user) and the server(tornado) in order to send the json message that has been created to our tornado server. Finally the visualisation of the results' is being rendered through the 2D figure and if the user has chosen a point, he will be able to see the protein structure through the JSmol.

The three layer connection between PELE,tornado server, pyProCT, and our GUI is as following:

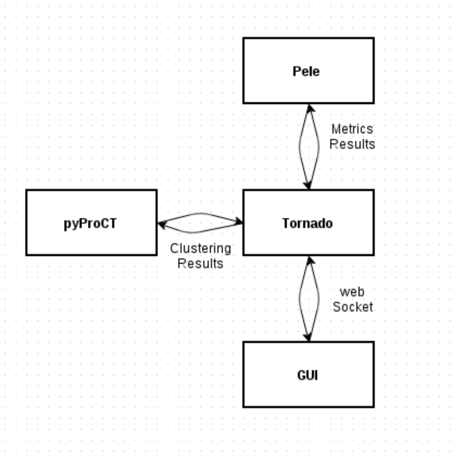


Figure 1: A diagram that describes how PELE, Tornado, pyProCT and our GUI are connected.

More specifically, the main idea is: A user can upload his protein and ligand in the PELE software. PELE will then create spatial positions(moving and rotating the protein and the ligand) and get metrics' results. At this point our visualisation procedure is being started.At the beginning, protein and ligand conformations are being represented as black points. In our implementation, after the user presses the button of the parameters window, the clustering of PyProCT is giving the results, and our GUI represents the clusters with different colours on a Voronoi diagram.

The clustering strategy, is giving us the opportunity to group and cluster our unlabelled data. Every cluster's objects represent a specific piece of information which are different from the objects that belong to another cluster. In this way, our results can be easily compared.

The Voronoi diagram is a partitioning of a given space into a number of regions. The colour differentiation of the cluster results is based on an algorithm using the HSL geometric cylinder.

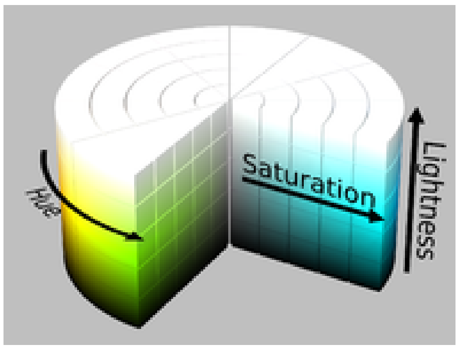


Figure 2: Using the HSL geometric cylinder, we manage to get different color shades.

Using its Hue, Lightness, and Saturation characteristics and setting upper and lower bounds, allows us to get colour shades that can be distinguished by the human eye. Lighter shades are being used for Voronoi's background.

A zooming semantic is being implemented for the zoom in and zoom out effect, so that the points can be successfully adapted and distinguished. Also note that the web page is being adjusted to the screen size each time as well.

Finally, having prepared a set of questions for our test users and using the eyes tracker, interviews were accomplished to improve the GUI. The workflow in our second video, presents tasks similar to what the users performed.

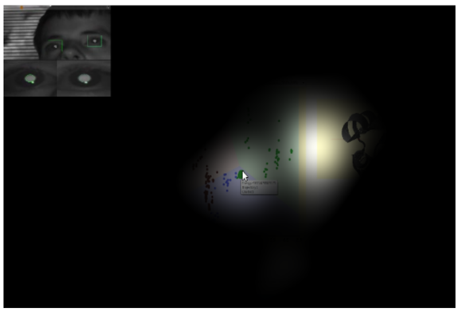


Figure 3: Eyes Tracker Opacity Map, catches the user's eye attention.

Results

Following are the results of our visualisation represented while a user navigates through our graphical interface.

At first the user is clicking the button in order to get results(painted points). Otherwise the diagram is represented with black points (bullets).

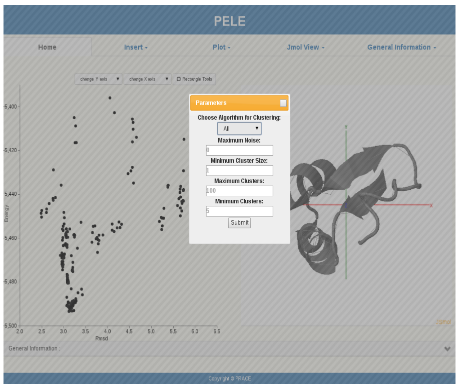


Figure 4: The basic dialog box in order to submit parameters.

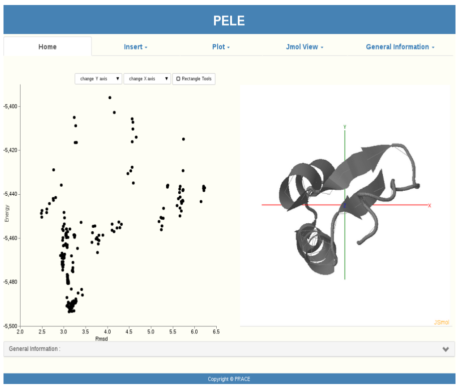


Figure 5: Without Parameters black points are returned since there aren't clustering results.

After he has submitted the parameters, he can then plot a protein with the One Clustering option. The user can change axis parameters, use the zoom in and zoom out, he also can plot another clustering and, by selecting a dot (protein), he can watch the protein's structure in the Jmol. The Rectangle option gives the user the opportunity to select more than one point(protein) in order to get information only for these proteins and, finally, to compare them.

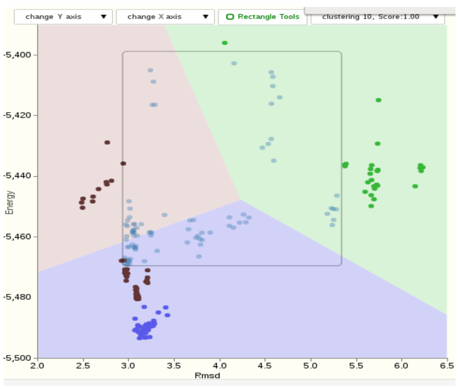


Figure 6: Rectangle tools in order to choose the points for clustering.

The Multiple Clustering option allows the user to plot all the clustering results in the same window. By double-clicking each plot, the plotted image appears larger in order to see further details.

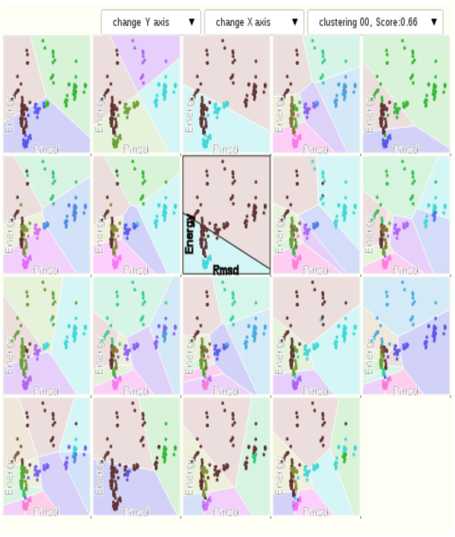


Figure 7: With the Multiple Clustering option, the user can see all the clustering in one figure.

Having selected one clustering, with the Multiple Axis option we can get all the possible axis combinations in one figure. The shape of the points can help the user compare results. With a double-click, he can also plot an image in a larger appearance.

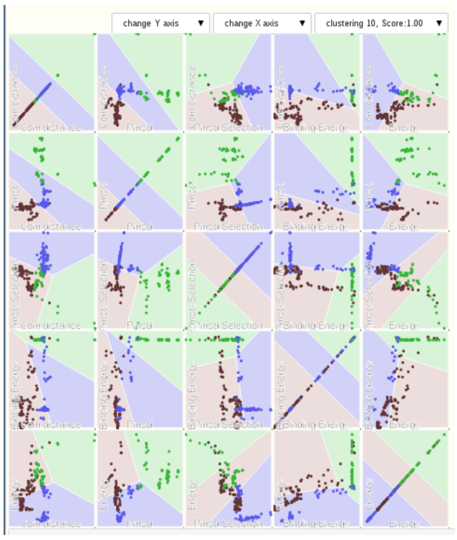


Figure 8: Multiple Axis option, gives all the possible axis combinations.

With the Bar Cluster option, clustering bars are represented with the scores of each clustering that has been given by PyproCT. A drop down menu gives the option to select different values such as Cluster Size, Noise etc.

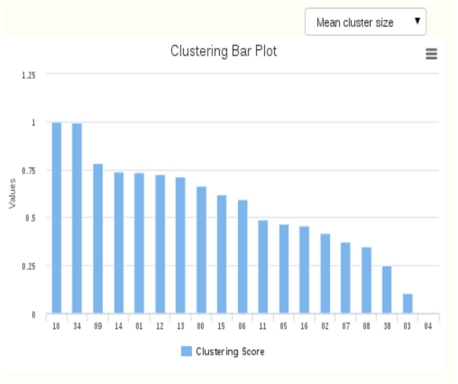


Figure 9: With the Bar Clustering Result the user is able to compare all the clustering by choosing a parameter.

Another useful tool is the JSmol appearance. The user can select between one, two, or none. If he selects the appearance of two JSmols, two icons are represented at each JSmols' title, in order to be informed of which dot has been selected in each. Furthermore, each JSmols' background gets the colour of the representative dot.

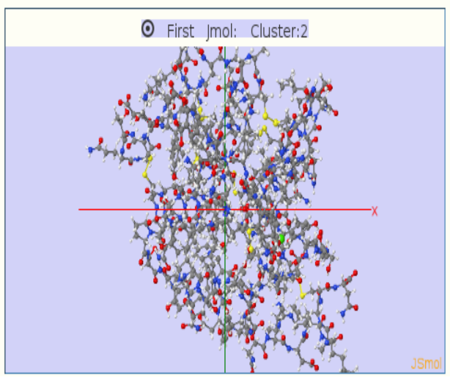
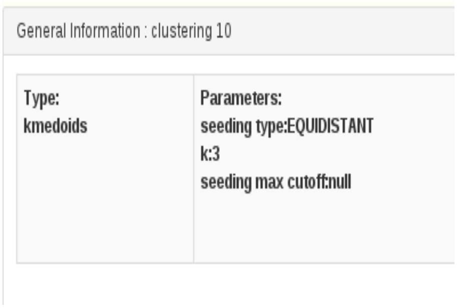


Figure 10: Jmol Appearance has the background color and icon in its title from the chosen point.

Finally, the user can select the General Info option for the selected cluster in order to get information about it, such as the algorithm that has been used, how many clusters it consists of etc.



General Information : clustering 10

Type:	Parameters:
kmedoids	seeding type:EQUIDISTANT k:3 seeding max cutoff:null

Figure 11: General Information give the user information about the parameters and the results

Note that for each button a Mousover effect triggers a message informing the user about this selection, every time he places a mouse over this area.

GUI Ready for Use

We have shown how to visualise, cluster, and evaluate PELE'S results by using a server, PyProCT's clustering methods, and a graphical interface. This visualisation provided more than a graphical representation of the proteins' structure by pointing on a specific "dot" (protein) or by selecting a whole area of "points"(proteins) with the "Rectangle" choice.

General information is given every time a user is interested in information, such as Evaluation, Number of clusters, etc. JMol View provides an optional view of more than one protein simultaneously. Even the plotting options are enriched with many functions.

Our GUI for PELE, is a user friendly graphical interface which is easily managed even by users with little experience in this field by using the tools and effects it provides, such as the Mouseover information triggering effect.

References

[1] Victor A. Gil and Victor Guallar (2014). pyProCT: Automated Cluster Analysis for Structural Bioinformatics. *J. Chem. Theory Comput*, 10:3236-3243.

PRACE SoHPCProject Title

Collaborative interface for in-situ visualization and steering of HPC molecular dynamics simulations

PRACE SoHPCSite

Barcelona Supercomputing Centre, Spain

PRACE SoHPCAuthors

Simos Kazantzidis, [National and Kapodistrian University of Athens,] Greece

Simos Kazantzidis

PRACE SoHPCMentor

Maria Ribera Sancho, UPC, Spain
Fernando Cucchietti, UPC, Spain

PRACE SoHPCContact

Maria, Ribera Sancho, UPC
Phone: +34 93 413 4035
E-mail: ribera@essi.upc.edu

PRACE SoHPCSoftware applied

Python, Node.js, Javascript, HTML, CSS, D3.js

PRACE SoHPCMore Information

http://d3js.org, http://nodeschool.io

PRACE SoHPCAcknowledgement

Author would like to thank his mentor as well as Jorge Estrada, Victor A. Gi and David Garcia Povedano for the IT support and Daniel Lecina, Luz Calvo Flores for their contribution to the initial version of the GUI.

PRACE SoHPCProject ID

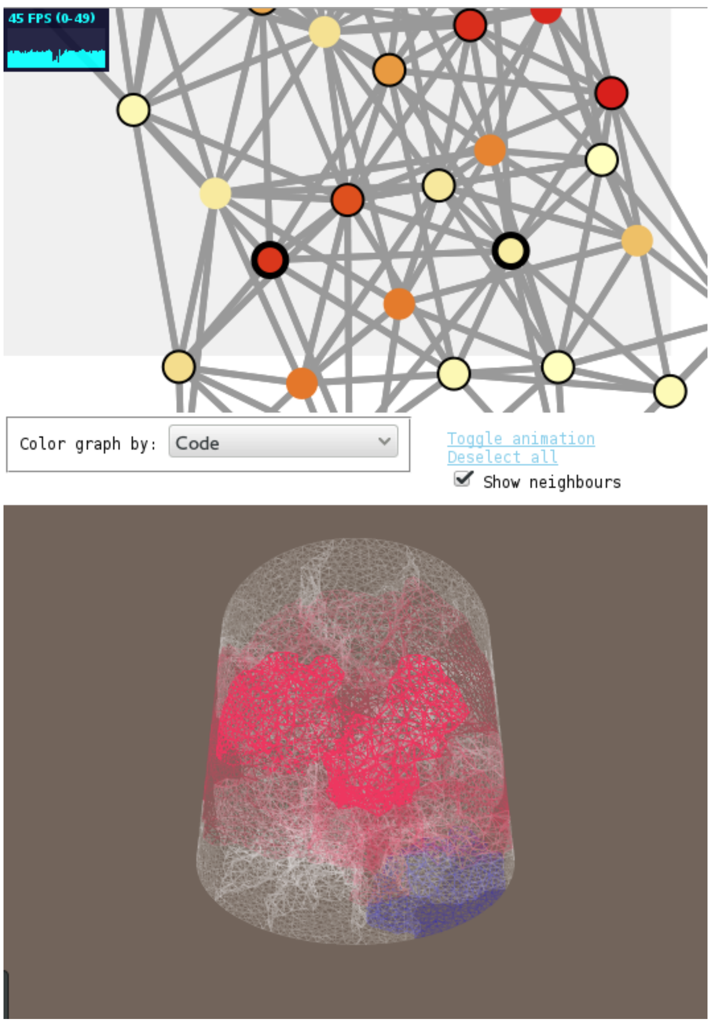
1501

Graphical interface for real time monitoring and automatic event detection in HPC parallel software

GUI for RT Monitoring in HPC

Rumyana Rumenova

Large HPC simulations can create local signs of failure or success long before a global signature is observed. If these signs are detected early, valuable computer resources could be spared.



Alya is a simulation system designed for large-scale research computations to be run on supercomputers. Initially used for computational mechanics simulations, it has expanded to other projects, including a large-scale simulator of the human heart. With the help of computational resources that have never before been available, researchers are able to better understand the inner workings of the human body at a level of detail much finer than was possible in the past. The work on Alya continues into other anatomical systems, to build a highly accurate model of the human body, which helps medical doctors diagnose pathologies, plan operations and test new drugs.

Developed at the Barcelona Supercomputing Center (BSC), Alya is used for several active projects to run simulations on Marenostrum, Spain's most powerful supercomputer. A supercomputer is really a room full of computers connected together through a fast network. Thus, running computations, or jobs, involves a complex partitioning of the problem into smaller sub-problems that can each be computed by one machine. Arriving at the final result requires tight coordination between these.

This process creates many difficult problems in achieving correctness and efficiency. One of these is that a mistake at the beginning of the computation may go unnoticed until the end, when results are aggregated - resulting in wasteful computation. As resources on a supercomputer are expensive, in terms of both the energy used and the support necessary, effort is often put into minimising their usage. Of course, researchers can also spend their time more fruitfully than by waiting for results.

The GUI in short

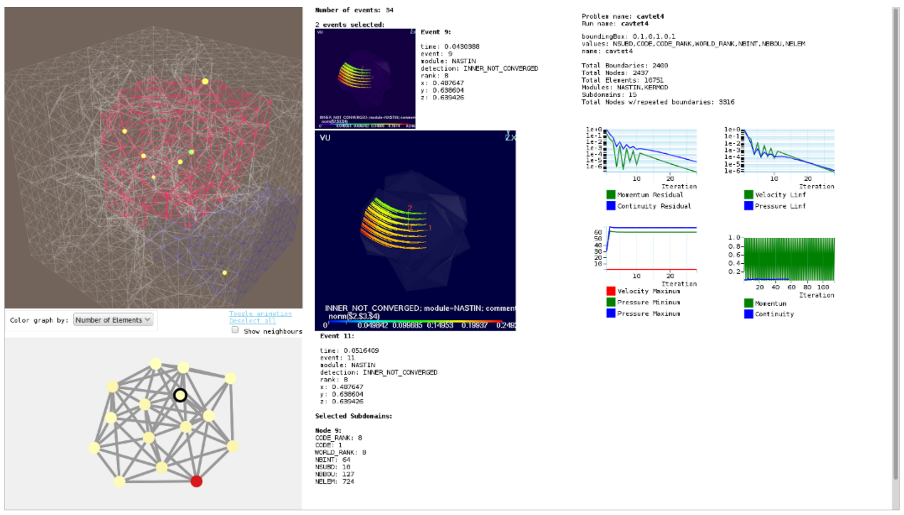
During this internship, I built a user interface which aims to help researchers manage their running jobs, evaluate the seriousness of any issues, and monitor performance metrics. It is based on BSC's real-time monitoring tool which finds potential errors in jobs running on an HPC cluster. My piece of the project will hopefully allow researchers to diagnose errors by interacting with visual representations of both the simulation problem (as a 3d rendering) and the connectivity between computing nodes (as a graph) on which it is running.

It can be seen at `http://www.bsc.es/viz/alyaevents/`

Usage

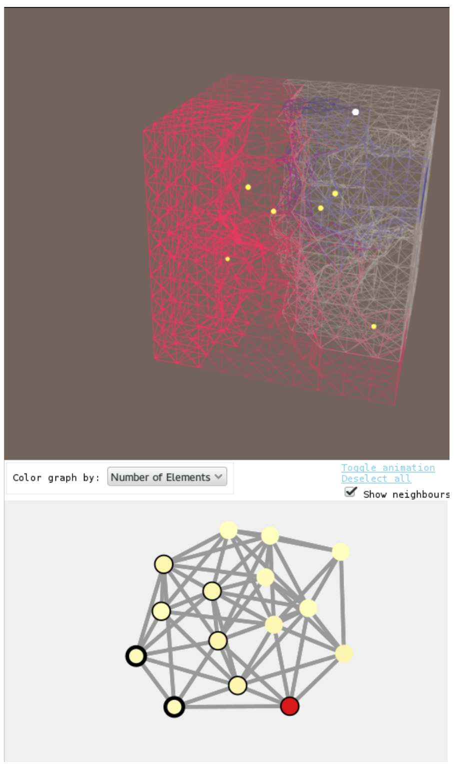
Users can navigate through different simulation tasks which have been submitted (*problems*) with several instances of the simulation per problem running on the supercomputer (*a run*). Some general information on the problem is always displayed, together with line charts which illustrate it.

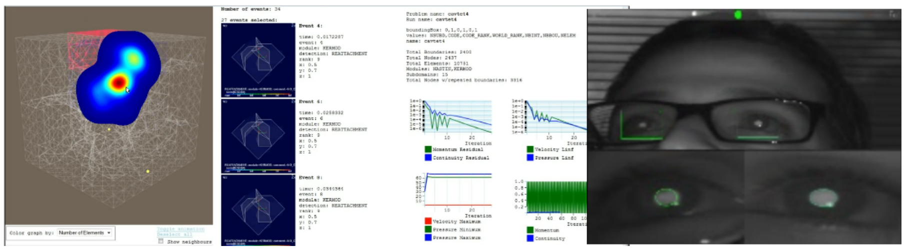
For each run, users can interact with a 3d representation of the object (mesh) that's being simulated, as well as a 2d graph illustrating the communication between separate subproblems (*sub-*



```
Number of events: 34

2 events selected:
Event 9:

time: 0.0490388
event: 9
module: NASTIN
detection: INNER_NOT_CONVERGED
rank: 8
x: 0.487647
y: 0.638604
z: 0.639426

Event 11:

time: 0.0516409
event: 11
module: NASTIN
detection: INNER_NOT_CONVERGED
rank: 8
x: 0.487647
y: 0.638604
z: 0.639426

Selected Subdomains:

Node 9:
CODE_RANK: 8
CODE: 1
WORLD_RANK: 8
NBINT: 64
NSUBD: 10
NBBOU: 127
NELEM: 724
```

```
Problem name: cavtet4
Run name: cavtet4

boundingBox: 0,1,0,1,0,1
values: NSUBD,CODE,CODE_RANK,WORLD_RANK,NBINT,NBBOU,NELEM
name: cavtet4

Total Boundaries: 2400
Total Nodes: 2437
Total Elements: 10751
Modules: NASTIN,KERMOD
Subdomains: 15
Total Nodes w/repeated boundaries: 3316
```





Top Left: The interface dashboard on a real problem run. Right: Neighbour selection in action. Bottom left: Results of user tests with eye-tracking camera.

domains), which may be running on one or several machines. A subdomain corresponds to one node in the graph and one region in the mesh. Users may select subdomains to find out more information on them, or discover subdomains which are exchanging messages with this subdomain (*neigbhours*) to arrive at the final computation output. This can be done through interaction with either the graph or the mesh.

Locations where a potential problem may have occurred (*events*), as detected by the real-time monitoring tool, are displayed within the mesh. Clicking on them shows the user more information on the problem, as well as an automatically generated image illustrating the event.

Functionality in detail

On the server side, the project is driven by a couple of scripts which find the available problems and runs, and then generate JSON objects to represent the connectivity graph. The outputs of these are read by the client application and displayed.

3d mesh - three.js

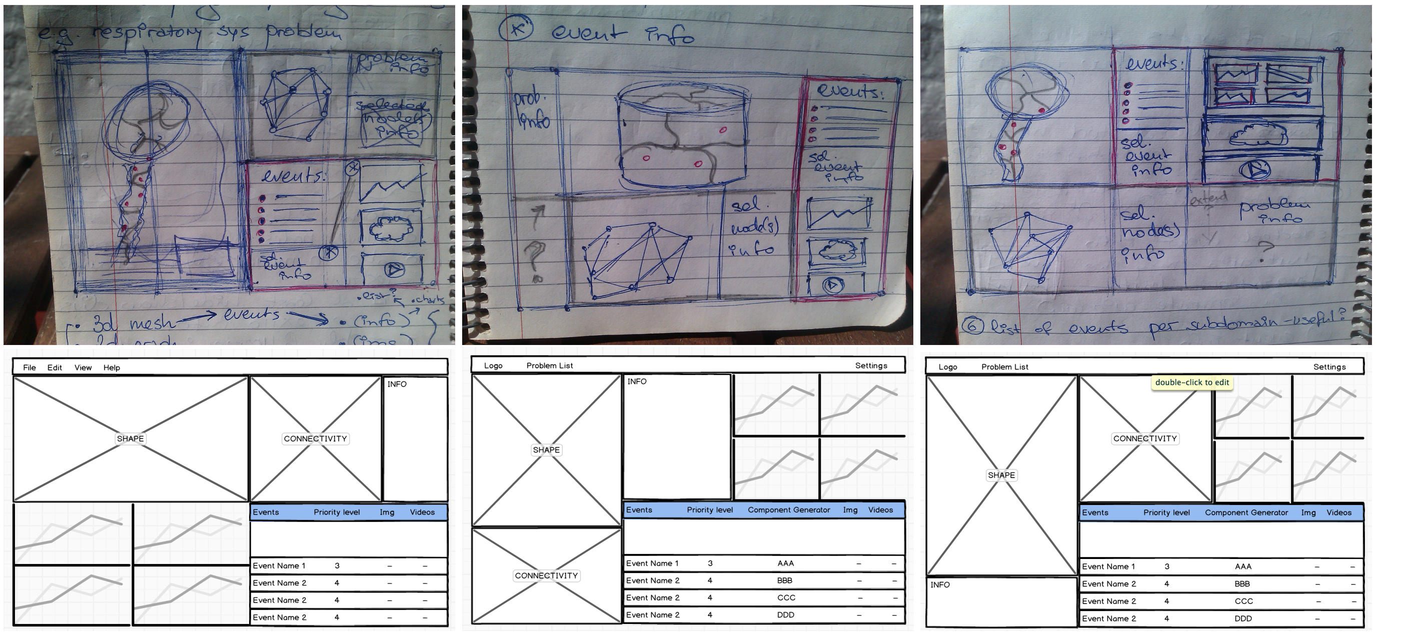
The simulation object is displayed as a wireframe to ease visibility. Subdomains are loaded as separate meshes and grouped into one when the application needs to make computations such as the object's bounding box, its center, etc. As images shown on the screen are 2d, a rendering involves a projection from 3d to 2d coordinates. For interaction with separate meshes, "unprojection" (raycasting) is used to find the 3d object corresponding to a cursor hovering over or clicking on a mesh or event. Changes in colour represent hovering or selection of a 3d element. Should the user choose to display them, neighbours are also highlighted using different opacities, on both the 3d mesh and the connectivity graph.

Mouse controls are used to allow the user to navigate the 3d mesh by rotating it or zooming in and out. A user may also enable animation which rotates the mesh around its y-axis, if they want to view it from all sides more easily.

Data on the 3d representation is read from STL files on the server. These are rendered on the client machine's GPU using Three.js over WebGL. Functionality is implemented as functions in *webGLplot.js*, allowing for code modularity such as one-line showing/hiding events, turning navigation and animation on/off, functions reused to implement "deselect all" functionality, etc. Some debug options (turned off by default) are also available, such as a rendering status screen which displays current frames per second and a small graph of recent graphics performance. Console messages are shown if a rendering is incomplete.

2d graph - d3.js

The connectivity graph is generated using a representation of its nodes and the communication between them. Colour is used to represent different characteristics, highlighting nodes which deviate significantly from the others - this makes it easy signal a master node to the user, or to let them examine the nodes where communication is heavier. Additional measures of this colouring can be implemented in the future. Selected nodes and optionally their neighbours are visually represented by a stroke around them, and changes over selections in the graph are reflected on their corresponding subdomain in the 3d mesh. Mouse controls enable zooming and panning of the graph.



Layout design sketches at different stages of development. Produced in order to ease user feedback, allowing to abstract away from functionality.

Graph visualisation is data-driven, drawing the graph from a simple JSON object, and dynamically choosing its colouring. Again, functionality (contained in *d3graph.js*) is modular, for example, including a "select subdomain" function which calls the appropriate "select node" from *d3graph.js* and "select mesh" *from webGLplot.js*.

Problem information - d3.js

As well as textual output, information is represented using line charts on various measures of interest. These are drawn dynamically by calling a function which takes as arguments the data path, measures to be displayed, colours, linear/logarithmic scale, and chart size. Their look and feel is specified in *style.css*. Charts can be generated out of multiple files containing different y-axis attribute values. Thus, they can be made more interactive in the future, adjusting their size or allowing the user to choose the attributes they want to see.

Methods

Our aim for this internship was to produce a minimal working version of the tool, with an accent on usability. I worked with the Scientific Visualisation Group in BSC, who have put a lot of thought into the best way to represent this type data, and in collaboration with the team that works on Alya, who will be the first to use the application.

In prioritising separate tasks, we followed user stories and consulted the prospective users in person. The project development followed an agile methodology, testing small incremental steps throughout, and evaluating usability continuously.

The tools used for this project are the standard web development HTML/CSS/JavaScript stack with extensive use of d3.js - a library for producing interactive 2d visualisations and binding DOM elements to data, and three.js for 3d, a library based on WebGL, a powerful visualisation API which achieves good performance by using the client's GPU for graphics rendering.

Evaluation

Experimental set-up

The final product was tested with 5 users, 3 long-term frequent users of Alya who are completely unfamiliar with the interface, and 2 infrequent users of Alya who are relatively unfamiliar with the interface. The test set-up consisted of nine small tasks, devised to be a realistic representation of what the user might want to do in their usual debugging process. The users' input was used to evaluate the experimental set-up: all tasks were deemed realistic by Alya users. Tasks were kept short so that they can be completed in 10 seconds or less by an experienced user of the interface.
Users were asked to talk aloud while they are completing the tasks, in order to record their current doubts and expectations.

A score of 0 was given when users were unable to complete the task in the time frame of about a minute. A score of 1 reflects that the user was almost able to complete the task, or was focused on the right layout element but needed some guidance to complete it. A score of 2 was awarded if the user was able to complete the task in less than 20 seconds, or had completed it already on their own during one of the previous tasks.

The tasks assignment was followed by an informal interview to find out the users' general opinion of the interface and their suggestions for improvement.

Tests were conducted using an eye-tracking camera to analyse eye movement, showing where the users expected to find a certain element on first, second or third trial, whether they searched for it across the screen, which locations were their main focus, and to compare their verbal testimonials with the reality of how they completed the task.

The following tasks were used:

1. Find the name of the current problem and the number of subdomains in it.
2. Pick an event.
3. Find the time it occurred at.
4. Find the subdomain it occurred in.
5. Find the neighbours of this subdomain.
6. Look at the mesh from the top.
7. Zoom in to the event you selected.
8. Find a subdomain which has a very different number of elements compared to the one you selected.
9. Approximately how many iterations have been done on this run?

Results and Discussion

Table 1: Experiment results: Task, User scores (1-5), Aggregate Score, Average Score

T	U1	U2	U3	U4	U5	Agg	Avg
1	2	2	2	2	1	9	**1.8**
2	0	2	0	0	1	3	**0.6**
3	2	2	2	2	2	10	**2**
4	1	0	0	0	2	3	**0.6**
5	0	0	0	2	1	3	**0.6**
6	2	1	2	2	2	9	**1.8**
7	2	2	2	2	2	10	**2**
8	2	1	1	1	0	5	**1**
9	2	2	2	2	2	10	**2**

Task 1. Problem information is differentiated well from the rest of the content, and has a suitable location.

Task 2. Points on the mesh representing events should be noted explicitly. Note that all users are new to the event system (have not used it so far). Frequent Alya users interpreted points as the geometrical centre of each subdomain mesh, as this is a representation they use.
Possible solution: Have a legend at the top of the 3d mesh.

Task 3. Event information is located suitably. It is labelled well enough for users to find a piece of information of interest straight away.

Task 4. Users search for this as explicit information. From looking at the mesh it is hard to tell which subdomain an event belongs to. Note, some confusion may arise from terminology: "rank" for example could be interpreted as an MPI_World rank.
Possible solution: Explicitly add relevant subdomain to event information.
Room for improvement: Information on the neighbours would be very useful to output as well.

Task 5. Users are prone to interacting with the mesh manually to find neighbours. They are neither able to see the "Show neighbours" option, nor inclined to look for it. Possibly they are interpreting it as part of the 2d graph menu.
Possible solution: Include it within 3d mesh area.

Task 6 and 7. Navigation is intuitive. Many of the users had already started rotating the mesh and zooming before this task came about.
Room for improvement: It would be useful to have the option of going back to original mesh position, and possibly some standard views (top, front, etc). Rotation axis could be surprising depending on the specific problem, as they change depending on information from the 3d mesh file.

Task 8. Some users were prone to looking at subdomain information to find it. A hint towards a different option led them to selecting a subdomain by colour.
Room for improvement: - A legend of the colours would be good for added understanding.
Colouring by other characteristics would be useful: specifically rank and code.

Task 9. Charts are easy to find and interpret even without a specific mention of using them for iteration information. This is also the most familiar element of the interface, as they have been modelled after a debugging tool that is already being used by Alya users.
Room for improvement: Some of the charts (the first 3) should be drawn differently to display outer instead of inner loop iterations.

Conclusion

The GUI developed over the summer is a minimal working version of a visual debugging tool which has been met with enthusiasm amongst Alya users. Layout positioning has been shown to make sense for users.

The application includes all the main elements of functionality as requested in initial user story documents and interviews. It is able to visualise different problems and does not exhibit prohibitive speeds under user interaction on Linux, Mac and Windows with recent versions of Firefox, Opera and Chrome.

As an active project, it will be improved in the future to better reflect what has been discovered through various user tests.

Acknowledgements

I would like to thank members of the Visualisation team who helped for this project in various ways - Fernando Cuccietti, Luz Calvo, David Garcia Povedano and Simos Kazantzidis, as well as members of the Alya team who provided valuable feedback - Guillaume Houzeaux, Juan Carlos Cajas, Ane Beatriz Eguzkitza.

PRACE SoHPCProject Title
Graphical interface for real time monitoring, automatic event detection, and alert triggering in HPC parallel software

PRACE SoHPCSite
Barcelona Supercomputing Center, Spain

PRACE SoHPCAuthors
Rumyana Rumenova, University of Edinburgh, UK

PRACE SoHPCMentor
Fernando Cucchietti, BSC, Spain

Rumyana Rumenova

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

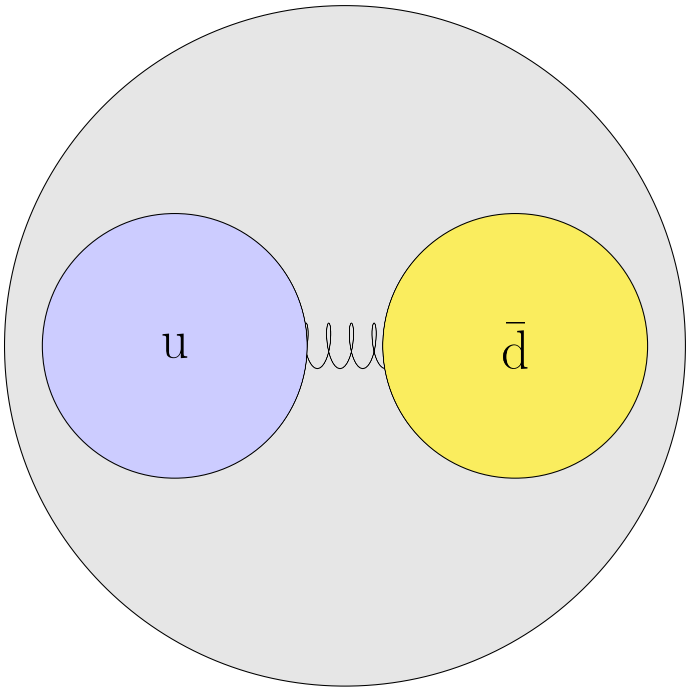
PRACE SoHPCSoftware applied
three.js/WebGL, d3.js

PRACE SoHPCMore Information
The project: www.bsc.es/viz/alyaevents
The Alya System:
www.bsc.es/computer-applications/alya-system

PRACE SoHPCProject ID
1502

Optimised I/O and Topological Susceptibility

Conor Larkin



The aim is to optimise I/O for eigenvectors produced by tmLQCD, using modern parallel I/O frameworks, and to write a measurement program for the topological susceptibility with the eigenvectors as input.

Lattice Quantum Chromodynamics is a field which has offered much to physics in recent years. This is due to both an increase in computing power and an improvement in the algorithms used. Something which has hindered it in the past is the huge expense involved in doing calculations at physical values of the up and down quark.

There are various methods to help decrease the cost of these calculations which allow us to get closer to these realistic masses. One which has had a lot of recent success is Deflation. During these lattice calculations, we are repeatedly required to invert huge sparse matrices. On these types of matrices, regular inversion is inefficient and we use methods such as conjugate gradient instead.

The time it takes for the Conjugate Gradient to converge to the correct solution is dependent on the square root of the ratio of the largest to smallest eigenvalue. The larger the ratio, known as the matrix condition number, the longer the algorithm takes. Deflation greatly speeds up this convergence by calculating a number of the lowest eigenvalues and projecting, or essentially removing them from the system that needs to be solved.

The central aim of my project is to improve the saving and reading (I/O) of these eigenvectors, so that it becomes efficient to store and reuse them for each calculation on the same matrix, instead of recalculating them every time. This will then be used in a program which will determine the topological susceptibility of a lattice configurations[1].

The first step was to implement the writing and reading functions for these eigenvectors using using raw binary files and MPI-IO, to serve as a baseline for further improvements. In tmLQCD[2], the lattice is split into local lattices which exist on each process. In this way, each process only has access to a section of each of the total number of eigenvectors. The eigenvectors and eigenvalues were written into one large binary file in parallel with all of the separate MPI process given an appropriate view of the file. This was significantly faster than the original implementation which required that each eigenvector be written in a separate file, with its own metadata and check sums. However despite its speed the raw binary implementation using MPI-IO has significant disadvantages, there is no metadata available and no check summing to protect against file corruption. For this reason, an alternative method was examined.

HDF5 is a data model, library, and file format for the managing and storing of data. Parallel HDF5 is a configuration of the HDF5 library allowing one to read and write to the same file across MPI processes. A HDF5 file can be thought of as more of a directory containing sub-directorys, known as groups, files, known as datasets, which allows us to store each eigenvector as a distinct dataset and attributes which are the metadata. This allows us to store information about the lattice size, the number of eigenvectors and such where it can be very easily accessed. HDF5 provides checks such as check summing which can help tell us when there is an issue with the data, such as it being corrupted. The idea behind using HDF5 is not to gain a performance boost over MPI-IO, but to add functionality and checking at a minimum cost. We wish for our HDF5 implementation to be as close in performance as is possible to the MPI-IO implementation.

The writing functionality of these 3 methods were tested a number of times on the Juqueen supercomputer on a 1024 node configuration, or one rack. The eigenvectors of a Dirac operator matrix of a $48^3 * 96$ lattice at the physical point of the pion mass were written to a file approximately 1 terabyte in size. The average time taken of each of the methods times was calculated and is shown in Figure 1 below.

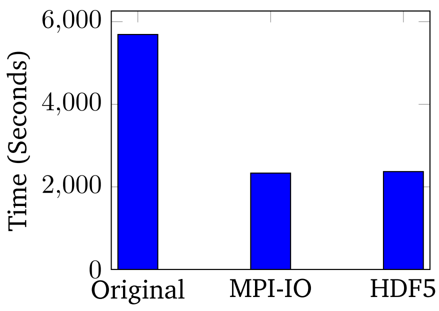


Figure 1: Average Write Times

As can clearly be seen there is a large speedup of approximately 2.5 in the MPI-IO and HDF5 versions over the original implementation. Also there the HDF5 write is only slightly slower than using pure MPI. This is as was hoped. The average read times were then evaluated in the same way and the results are shown in Figure 2.

The MPI-IO is faster than the original by almost a factor of 6 which is great but we see our first issue with the HDF5 implementation. We known that HDF5 should mirror the performance of MPI-IO so we reason that it must be caused by something different about the structure of the files and in fact it is. In the MPI implementation eigenvectors and eigenvalues are stored in one large memory block but in the HDF5 version, while it is in one file, each of the eigenvectors are stored in different datasets. While storing them in different datasets makes it easier to read a specific subset of these eigenvectors in a file, it is still possible to do this when they are stored as a block thanks to the dataset slicing functionality in HDF5.

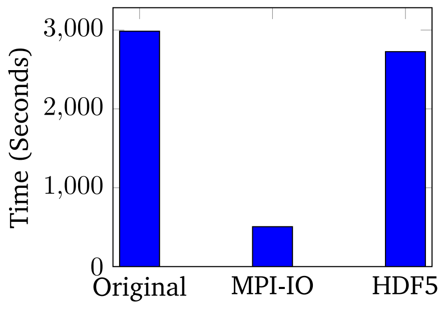


Figure 2: Average Read Times

This is considered a reasonable trade-off if it gives us the speed up that will get us close to the MPI read time. I implemented the block HDF5 method and the results for the average read time are shown below in Figure 3. The write time for the block and non-block HDF5 methods are virtually identical

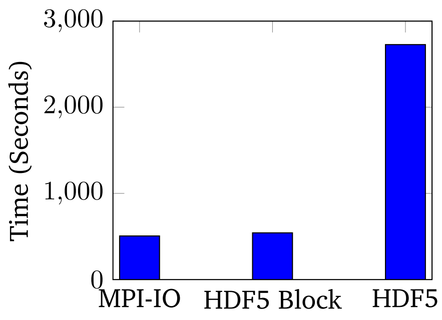


Figure 3: Improved Average Read Times

We have gained an approximate 2.5x speedup in the write times and a much larger one in read times. We have achieved what we set out to do in terms of the I/O section of the project.

The vacuum of Quantum Chromodynamics has a topologically non-trivial structure. Certain properties of QCD require that the ground state be a superposition of gauge configurations with different topological properties. The topological susceptibility allows us to measure the topological fluctuations of the vacuum.

A lot of people will have heard of the Higgs Mechanism which gives mass to certain Standard Model particles through spontaneous symmetry breaking. There are in fact other symmetry breaking mechanisms which give different particles mass and topological susceptibility calculations can give us insight into these processes. In particular topological susceptibility has been shown to have an important effect on the mass of the η' meson. The topological susceptibility χ_t is defined as,

$$\chi_t = \frac{\langle Q_t^2 \rangle}{V} \qquad (1)$$

where V is the volume, Q_t is the topological charge and $\langle . \rangle$ denotes the average. The topological charge is mathematically quite complicated so I will attempt to give an intuitive introduction to topological equivalence.

Two field configurations are said to be topologically equivalent if it is possible to topologically deform one of them continuously into the other without passing through forbidden field configurations which are unphysical.

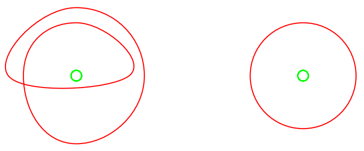


Figure 4: Topological Example

Above is an example of topologically in-equivalent systems. From the point of view of the green points, it is impossible to smoothly transform one system into the other. Assuming theses curves travel anti-clockwise these systems can be said to have a winding number of 2 and 1 from left to right. Topological charge can be seen as something similar, an integer number, in which systems with the same topological charge are topological equivalent and those with distinct ones are topologically in-equivalent.

I have written a program which will calculate the topological susceptibility of a gauge configuration using the eigenvectors provided by the tmLQCD package and read through the HDF5 Block read function.

References

[2] L. Giusti and M. Luscher, JHEP **0903** (2009) 013 [arXiv:0812.3638 [hep-lat]].

[1] K. Jansen and C. Urbach, Comput.Phys.Commun. 180, 2717 (2009), arXiv:0905.3331 [hep-lat]

PRACE SoHPCProject Title
Optimal deflation in the linear solver for lattice QCD

PRACE SoHPCSite
The Cyprus Institute

PRACE SoHPCAuthors
Conor Larkin, [Trinity College Dublin,] Ireland

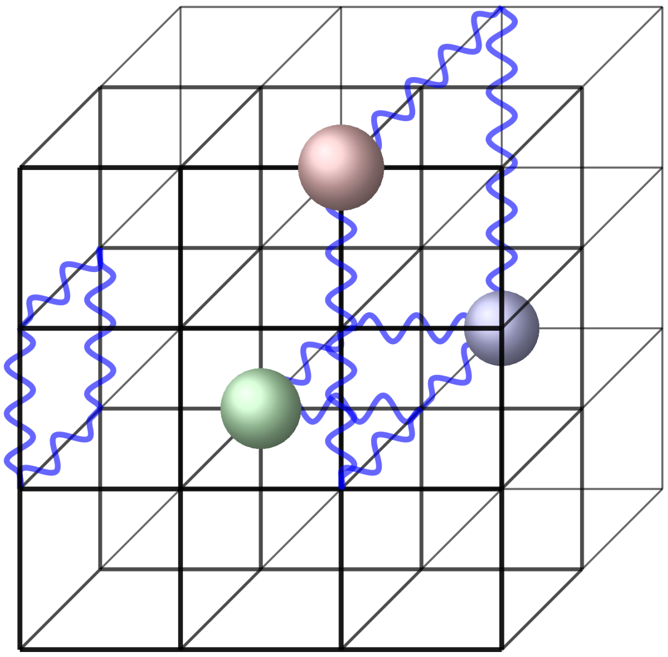
Conor Larkin

PRACE SoHPCMentor
Giannis Koutsou and Constantia Alexandrou, The Cyprus Institute, Cyprus

PRACE SoHPCProjectID
1503

PRACE SoHPCAcknowledgement
I'd like to thank Giannis Koutsou, Constantia Alexandrou, Abdou Abdel-Rehim and Jacob Finkenrath for all their help and guidance.

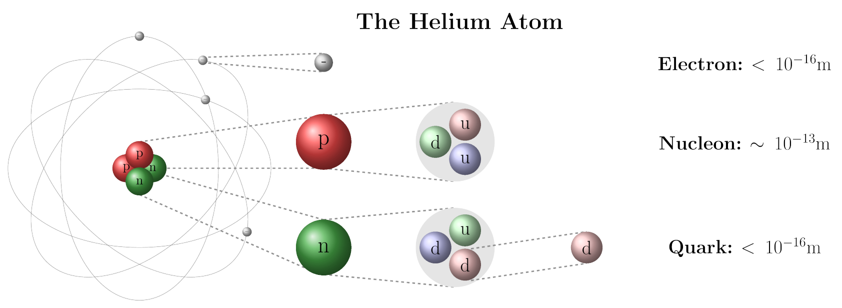
Hybrid High Performance Computing in Lattice QCD



Simon Pfreundschuh

Lattice quantum chromodynamics computations put extremely high demands on computing power. To meet them, getting the most out of todays supercomputing architectures is absolutely necessary.

Quantum chromodynamics (QCD) is the physical theory describing the behaviour of quarks and gluons. Those fundamental particles constitute some of the major building blocks of the world around us. There are six different quarks and eight different gluons. Combined in different ways, quarks and gluons make up all nuclear matter such as the neutron and the proton. Together with electrons, those form atoms, which again combined in uncountably many ways make up all the matter in the universe. The force that makes quarks come together to form new particles is called the strong force or strong interaction. The gluons are the mediators of the strong force. That means that they transmit the strong force between quarks. This is similar to the photon, which transmits electromagnetic force between charged particles. Coming to an understanding of the wold of subatomic particles was and is one of the biggest challenges of modern science. This challenge is met by experimentalists and theorists all over the world trying to advance the frontiers of physics. The underlying theory, however, has become so complex that some of the worlds most powerful computers are necessary to make predictions from it. Lattice QCD is a computational framework to predict physical quantities from the equations of quantum chromodynamics. Since those equations are too complex to be solved analytically, one has to resort to numerical approximations. In lattice QCD quarks are placed on a 4-dimensional lattice of points in space and time. The gluons live on the links between the lattice sites. In order to obtain realistic, physical results, the lattice has to be made finer and finer. Increasing the size of the lattice, however, also increases the computational complexity. While a vector describing a lattice of size $16 \times 16 \times 16 \times 32$ has about 1.5 million elements, a vector describing a lattice of size $64 \times 64 \times 64 \times 128$ already has about 400 million elements, requiring 6.44 GB of memory to store. Lattice QCD computations not only involve thousands of such vectors, but also matrices of this size. The computationally most demanding task in lattice calculations is the solutions of linear systems of the form $\mathbf{Dx} = \mathbf{b}$. The matrix $\mathbf{D}$ is a discretized version of the Dirac operator, a square, sparse matrix of size 12 times the lattice points. In a typical lattice calculation, thousands of such



systems need to be solved. This computation can be sped up with a technique called deflation. This technique require the computation of eigenvectors of the Dirac operator. Eigenvectors are special vectors that do not change their direction when acted upon with the Dirac operator. Such computations can of course not be run on ordinary computers, but require the highly specialised high performance systems.

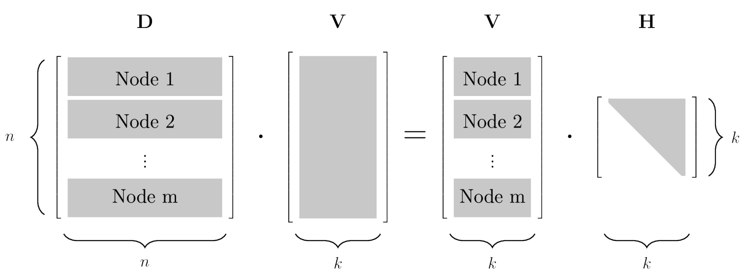


Figure 2: The computations of a Schur decomposition can be parallelised by splitting up the matrices column-wise over the compute nodes.

Most supercomputers today are built as commodity clusters. That means that they are made up of a very large number of compute nodes built from off the shelf components. Each compute node is basically a computer on its own, with its own multi-core CPU and memory. To further increase performance, the compute nodes are often equipped with one or more coprocessors or GPUs. Those are basically specialised hardware components for that speed up certain tasks. GPUs for example yield performance benefits for highly parallel tasks. The compute nodes are connected to each other over a high speed network. The general structure of such a hybrid cluster machine is displayed in figure 1. When running a program on a cluster machine, each node runs its own independent instance of the program. The independent instances of the program communicate with each other over the network. To achieve maximum performance on a cluster machine it is necessary to write a program that can run efficiently on many nodes, but also makes good use of the computational resources of each node. This requires a program that is not only parallel on node level but also exploits the parallel computing capacities of the nodes. Such a combination of different programming paradigms is generally referred to as hybrid programming.

The implicitly restarted Arnoldi method, which is used for the computation of the eigenvectors, is based on the construction of a partial Schur decomposition of the Dirac operator $\mathbf{D}$, which takes the form

$$\mathbf{DV} = \mathbf{VH} \tag{1}$$

where $\mathbf{H}$ is a $k \times k$ upper triangular matrix, $\mathbf{V}$ is a $n \times k$ hermitian matrixKLK and k is just the number of eigenvectors that we want to compute. This is an important point, because while we cannot work with $\mathbf{D}$ explicitly due to its size, the matrices $\mathbf{V}$ and $\mathbf{H}$ are actually of manageable size. All we need to know of $\mathbf{D}$ is how it acts on a vector $\mathbf{x}$. So there are two kinds of operations that need to be performed: The ones involving the Dirac operator $\mathbf{D}$ and the ones involving only the smaller matrices $\mathbf{V}$ and $\mathbf{H}$. Those operations can be parallelised, by splitting up the matrices $\mathbf{D}$ and $\mathbf{V}$ over the compute nodes. Then, each compute node only has to perform the computations corresponding to a sub-range of the columns of the matrices as is displayed in Figure 2. Due to the inherent parallelism of the application of the Dirac operator $\mathbf{D}$ to a vector $\mathbf{x}$, this operation can be very efficiently parallelised on a GPU. This was also already implemented in the lattice QCD code used in this project. The operations involving the $\mathbf{V}$ and $\mathbf{H}$ matrices, however, were still running on only one of the CPUs cores. The aim of the first part of my Summer of HPC project was therefore to parallelise also those computations.

To this end, I explored two different approaches to parallelise the eigenvector computation on the nodes. The first one was to link ARPACK, which is the software package implementing the implicitly restarted Arnoldi method, to a multi-threaded linear algebra library. Modern software is written in a highly modular way in order to increase the re-usability of code. Frequently used functionality is provided by software libraries. This has the advantage that not every time someone wants to perform linear algebra operations he has to write his own routines for that. Optimised and parallel linear algebra libraries are readily available in modern cluster machines programming environments. Apart from being quick and easy to realise, this approach has the advantage of minimising the risk of introducing new errors in the code. The second approach was to rewrite the ARPACK code in the hope to achieve greater performance gains. The idea behind this was that the splitting up of the $\mathbf{V}$ matrix can be extended to the multiple CPU-cores of each compute node. That means that the column sub-range of each node is once more split up over the CPU-cores of the node. The splitting up of the matrices over each nodes GPU and CPU cores is illustrated in Figure 3.

To assess the performance of the the two approaches, several benchmarks were performed. The first benchmark was performed on a development machine here at The Cyprus Institute, named Prometheus. Each of Prometheus nodes has two GPUs and a 12-core CPU. The benchmark run on Prometheus was a rather small one with a matrix $\mathbf{D}$ of size $n = 90000$ and the number of eigenvectors $k = 100$. It was run on only one node. The results are displayed in Figure 4, which displays the execution time of the program normalised with the serial execution

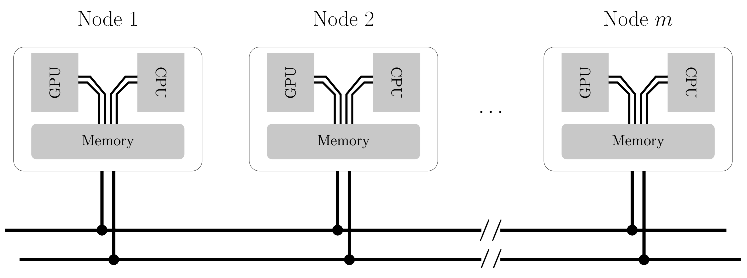


Figure 1: Modern supercomputers consists of a large number of compute nodes, each equipped with a multi-core cpu as well as one or more GPUs or coprocessors.

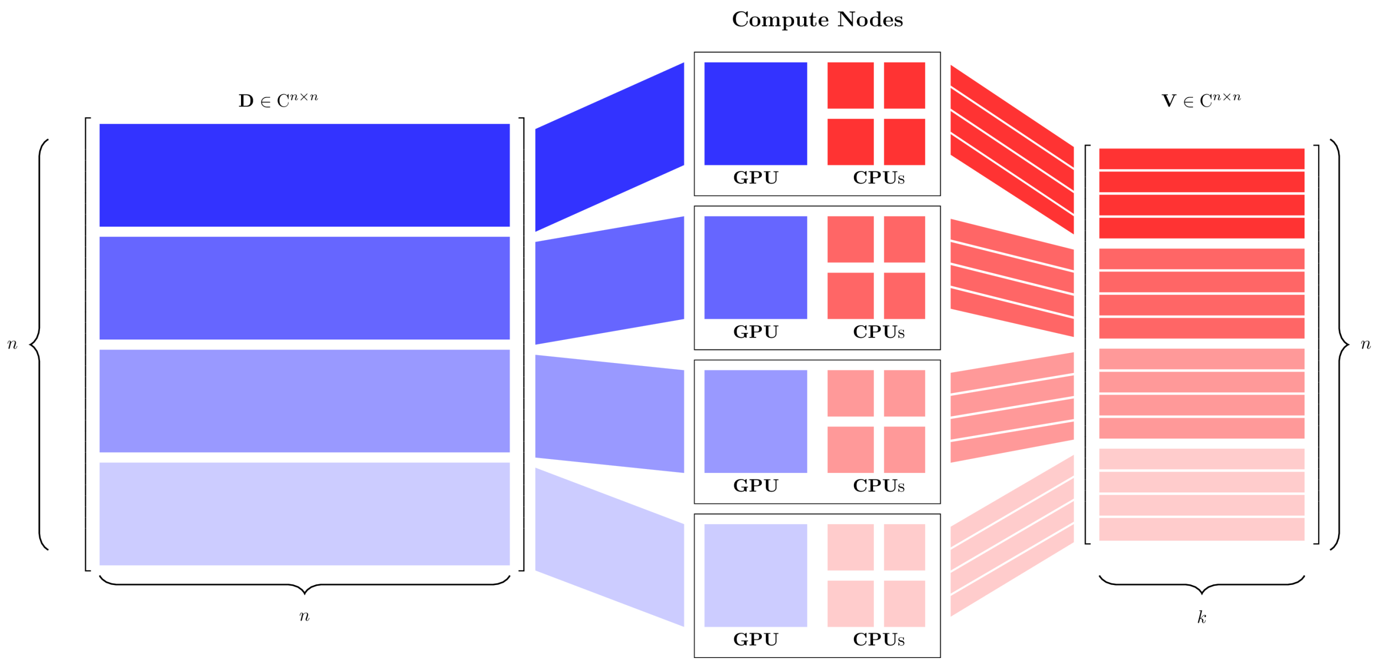

Compute Nodes
$\mathbf{D} \in \mathbb{C}^{n\times n}$
$\mathbf{V} \in \mathbb{C}^{n\times n}$
GPU
CPUs
n
n
n
k

Figure 3: The matrices involved in the computation of the eigenvectors are split up column-wise over the nodes to parallelise the computation.

time, i.e. the time the non-parallel version of the code takes to execute. On Prometheus the parallelised code actually performs slightly better than the code using multi-threaded linear algebra and speed-ups of a factor up to three using all twelve cores were reached.

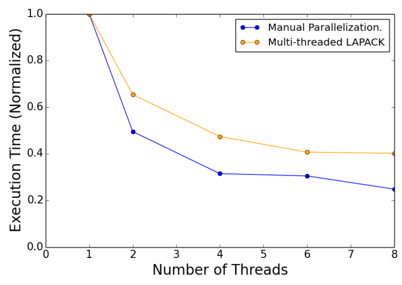


Figure 4: Benchmark results on Prometheus.

The second benchmark was performed on Piz Daint which is a hybrid supercomputer located at the Swiss Supercomputing Centre in Lugano, Switzerland. Each node on Piz Daint has an 8-core CPU and one GPU. The benchmark performed on Piz Daint was for a $48 \times 48 \times 48 \times 96$ lattice and run on 64 nodes. The number of eigenvectors computed was chosen to be 1000. As displayed in Figure 5 the results obtained are not as good as the ones obtained on Prometheus. Both versions of the code scale worse on Piz Daint and the parallelised code has no advantage over the code using linear algebra.

Ideally, the expected gain in performance should be proportional to the number of cores. Even though this could not be achieved, the performance gain on each platform was at least 100%. The results also show that the manually parallelised code performs only slightly better than the code using multi-threaded linear algebra. The reason for this is that computationally most complex operations of the eigenvector computation are actually the linear algebra operations. Their execution time grows fastest with increasing matrix size and thus determines the performance of the overall program.

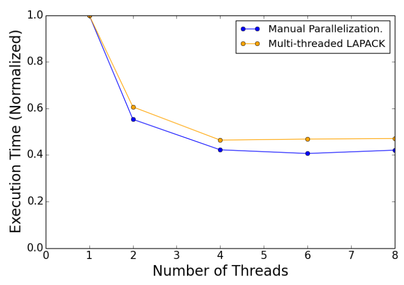


Figure 5: Benchmark results on Piz Daint.

The development process of this project was of course not as linear as presented in the report. Writing, compiling and running software on high performance computers are highly complex tasks and require great amounts of specialised knowledge. Different architectures and different available software libraries strongly influence the performance of software. Therefore, the above results can probably be further improved, but due to time restrictions we decided to move on to the second part of my project. The aim of the second part is to use the eigenvectors computed in the first part to improve the accuracy of the approximate inversion of the Dirac operator. I think that this project illustrates in a fascinating way the challenges in lattice QCD. Computations in lattice QCD not only require a thorough understanding of the underlying physics and mathematics, but also highly specialised programming skills that can get a maximum of performance out of modern supercomputers.

PRACE SoHPCProject Title
Optimal Deflation in the Linear Solver for Lattice QCD

PRACE SoHPCSite
The Cyprus Institute, Nicosia, Cyprus

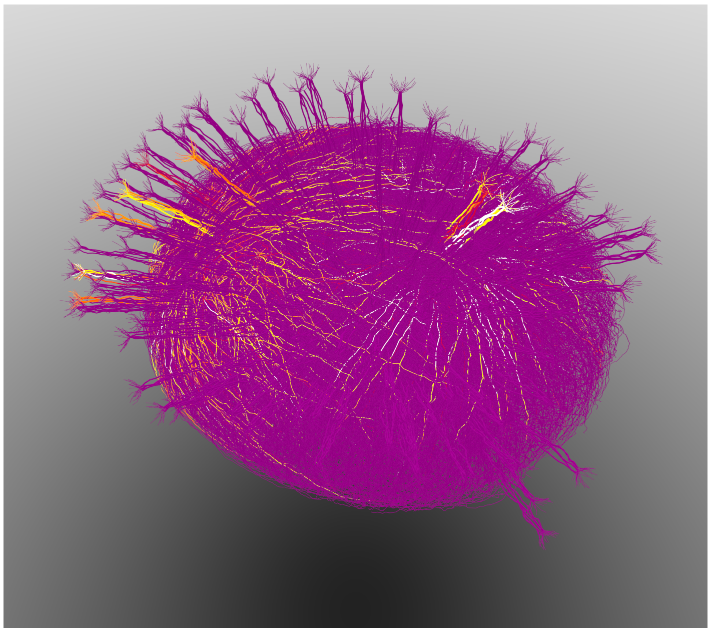
Simon Pfreundschuh

PRACE SoHPCAuthors
Simon Pfreundschuh, Chalmers University of Technology, Sweden

PRACE SoHPCMentor
Prof. Constantia Alexandrou , The Cyprus Institute, Cyprus Giannis Koutsou, The Cyprus Institute, Cyprus

PRACE SoHPCAcknowledgement
I would like to express my gratitude to Prof. Alexandrou and everyone in the lattice QCD group for welcoming me here at CaSToRC. Especially I would like to thank Giannis Koutsou, Abdou Abdel-Rehim and Jacob Finkenrath for all their help and knowledge shared with me. It was a truly inspiring experience. I would also like to thank everyone involved at PRACE for making the Summer of HPC program possible.

PRACE SoHPCProject ID
1504



Visualisation tool for Olfactory Bulb

Cem Benar

Contributing to finding out how odour discrimination is done in olfactory bulb and investigating functional properties of olfactory bulb neurons by providing a 3D visualisation tool.

3D visualisation tool is built to help find out how olfactory bulb neurons contribute to odour recognition in the human brain. When given the data of olfactory bulb neurons, the tool builds a three-dimensional view of the olfactory bulb neurons: mitral cells and granule cells. In addition, it enables researchers to analyse the olfactory bulb data in a visual and interactive way.

Introduction

The sense of smell results from the stimulation of receptors by molecules in the air. These chemical molecules reach the nasal epithelium during inhalation. These stimulants are transduced into electrical activity in the olfactory receptor neurons, which then transmits them to the olfactory bulb and from there to the rest of the central nervous system. Olfactory bulb is responsible for transmitting olfactory information into a number of areas in the brain. Its output is not a passive reflection of its input. It transforms odour input to help the rest of the central nervous system in discriminating the odour.

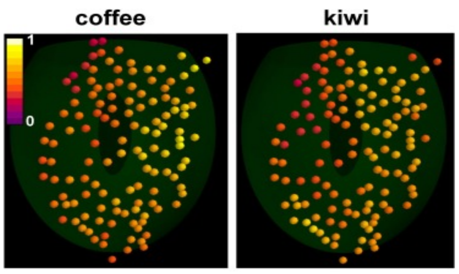


Figure 1: Glomeruli (coloured spheres) activation patterns[1]

Glomerulus (plural glomeruli) are spherical synaptic structures, which are found in-between olfactory receptor neurons and mitral cells. As seen in Figure 1, different inputs create different intensities on glomeruli. This helps researchers in making odour discrimination.

Mitral cells are the one of the main neurons' types located in olfactory bulb. They receive odour input from at least four cell types: olfactory receptor neurons, periglomerular neurons, external tufted cells, and granule cells. They help in transmitting olfactory information into the brain by processing in a way which is currently unknown. A mitral cell output is not a passive reflection of their input from the olfactory receptor neurons and external tufted cells. The exact type of processing that mitral cells perform with their inputs is still a matter of controversy. A mitral cell consists of tufts, apic, and soma and lateral dendrites. The connections between mitral and granule cells are made through the lateral dendrites. Tufts are a part of a mitral cell that takes the electric pulse from other neurons. The electric pulse coming from other neurons is transmitted to the apic through tufts. The apic is single cylindrical structure that connects tufts to the neuron cell body. It gets the electric pulse from tufts and transmits it to the cell body, which is called soma. Soma is the centre of a mitral cell. It transmits the electric pulse to dendrites. Dendrites are structures that enables a mitral to convey the information to granule cells. The geometry of a mitral cell is represented in Figure 2.

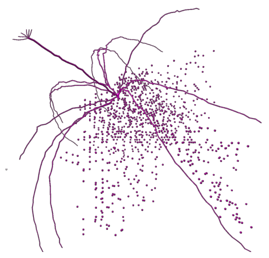


Figure 2: A single mitral cell with granule cells

Granule cells are another type of neuron cells located in the olfactory bulb and has important functions in odour discrimination like mitral cells and glomeruli have.

Mitral cells create a unique pattern for each type of odour input such as coffee and kiwi, while they process odour input. Similar smells produce similar patterns in mitral cells. These patterns become more unique and more easily identified as time goes on. With the help patterns of firing neurons, how odour recognition works in the olfactory bulb, especially in mitral cells and granule cells, is attempted to be understood. The motivation of the project is to develop a 3D visualisation tool to help researchers in their studies of odour recognition. If the whole structure is visualised, it is easier to find the patterns and classify them in order to understand which odour input is currently performed. This work provides researchers a framework as well as an application.

The design of the neural network is made by an open-source neural network simulator called NEURON. The simulator provides us synthetic data such as the topology of mitral cells and granules as well as their timing values. The binary data is capsuled as radius values and 3D points in the Cartesian coordinate system for the parts of the neural network which are tufts, apics, somas, dendrites, and granules. The data of tufts, apics, somas, and dendrites is more complex than granules' data. Tufts, apics, somas, and dendrites consist of substructures which we call segments. Each segment is defined in a data format of [(x1, y1, z1, r1), (x2, y2, z2, r2)], where x, y, z are axises in the Cartesian coordinate system and r is the radius. NEURON simulates the neural network in 40 seconds and generates the time values for dendrites and granule cells. Time values represent at which time during the simulation these parts have the electric pulse.

Working Environment

We used Blender 2.75 software as a 3D visualisation and an animation program. It is open-source and has many features including video editing. It is also good at rendering with a high quality. In addition to this, Blender allows its users to make modifications on its user-interface. Although Blender is generally used for modelling, with this work, I believe we have proved the ability of Blender in making a scientific visualisation with big data. In addition to the interactive user interface of Blender, it also offers programmers an embedded python console where you can make your work by using either user interface or writing python scripts.

Rendering Stage

Firstly, we started to render the parts of a mitral cell by using two separate approaches. The first approach is to draw a circle and extrude it to make a segment. The extruding approach is a natural way of rendering things, especially neurons. The second one is to define each segment of apic, soma, tufts, and dendrites as a cone object in Blender. Instead of creating a circle and extruding it, we directly created a cone structure. The reason to use a cone instead of a cylinder is that dendrites and tufts are in the shape of a truncated cone. Their radius values vary from segment to segment. Both approaches are implemented as two separate rendered classes. They are able to work in the same rendering case. In other words, each part in the neural network, which are apic, tufts, soma, dendrites and granules can be rendered by selecting either of them.

On average, each mitral cell has 600 segments and the number of granule cells related to that mitral cell is 1100. The neural network is made of 635 mitral cells and around 700,000 granule cells. The structure, in total, has 1,080,000 segments, if each granule cell is also counted as a segment. First of all, an object is created for each segment to render. For this reason, the first issue that is faced in Blender was the scaling problem. Creating objects in this huge number made the rendering process very slow.

The code was getting slower as the number of objects in the scene are increasing. Blender has operators (bpy.ops) for the users using the graphical user interface. It is not advised to use these operators in python scripts. For this reason, we changed our approach and started using bmesh module. Bmesh module offers more flexibility and features to render meshes. It is designed for those who want to create meshes by using programming. A mesh is a collection of vertices, edges, and faces that describe the shape of a 3D object. It is also called as object data. This approach improved the speed of rendering however; it did not solve the scaling problem. For example, rendering a hundred of mitral cells in the supercomputer of CINECA were not terminating in ten hours.

Then, the reason of the scaling problem was found out. When adding a new object with a mesh into the scene, updating the scene is internally done. Updating the scene means when a new mesh is added, Blender checks every time if each mesh in the scene is modified or not. As the number of meshes in the scene is increasing, the number of objects to be updated is also increasing, which causes slowness. For this reason, we decided to decrease the number of objects in the scene. Segments in each part of a mitral cell are grouped. In this way, creating a single object and mesh for each part of a mitral cell became possible by adding the geometry of each segment into the single mesh iteratively.

With this approach, we have 4 groups for tufts, apic, soma, and dendrites for a mitral cell. It solved the scaling problem in a very efficient way. Instead of creating 1,080,000 objects, 2541 objects and meshes which is 635x4 + 1 (one for all granules) are created for the whole neural network. Surprisingly, we realised that rendering the whole neural network doesn't need processing power of a supercomputer. For this reason, we did not feel the need to make the code parallel at this stage. Besides, Blender has a feature that automatically runs the code parallel. Either you can simply specify number of threads to be run or Blender decides it by itself by analysing to current power of hardware. Rendering 635 mitral cells represented in Figure 3 takes a few minutes with this approach in the supercomputer of CINECA.

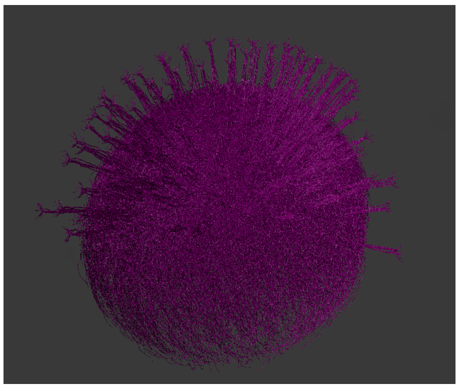


Figure 3: Olfactory bulb mitral cells

Colouring Stage

The second stage of the project was to colour the neural network in order to show electric pulses mitral cells have during the some intervals of the simulation.

We have thirteen different types of colours for this purpose. Colours represent strengths of an action potential. An action potential can be considered as an electrical signal a neuron carries. The essential colours are purple, dark pink, red, orange, yellow and white. The strength of an action potential is increasing in this order of colours. For example, purple segments represent the segments that have no electric pulse. White segments show the centre of currently moving electrical signal.

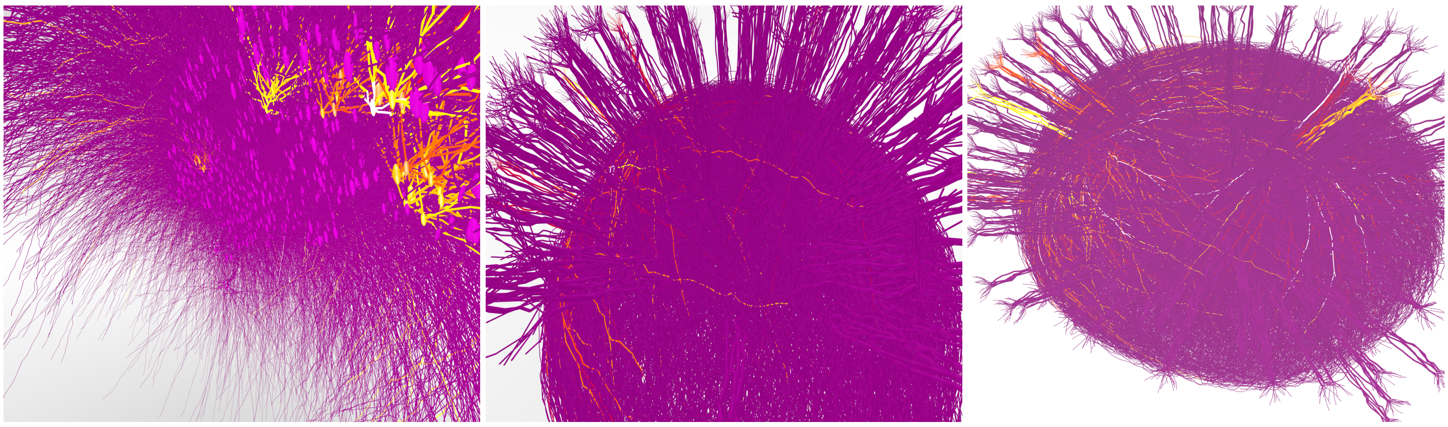


Figure 4: Olfactory Bulb with odour input

The time data is produced by NEURON for every segment of dendrites and granule cells, and somas but not for apics and tufts. For this reason, fake time data for apics and tufts are created.

At this point, the second important problem was come out. We were not able to assign a colour for each segment, since complex objects, including many of segments, are created. When a colour is assigned for an object, for example the dendrites of a mitral cell, all segments in the dendrites were becoming the same colour.

For this reason, a type of texturizing called UV mapping is implemented. UV mapping is considered as NxN matrix, where N is texture size. It is the representation of an object in 2D space. Every element of the matrix is called as a pixel. The pixels refer RGB values of the segments. When changing a pixel colour, the colour of a segment which is referred by that pixel, is automatically changing. With this approach, we succeeded to make colouring for the complex objects as seen in Figure 4.

Animation

We setup the duration of the animation 200 minutes for the simulation lasting 40 seconds to show the electrical propagation segment-by-segment.

During the animation, we faced with speed problems since there are many segments that are needed to be changed their colours in the scene. For this reason, a caching mechanism is implemented to make the animation in a faster way. The speed of transitions between frames became fast enough with the cache.

Adding New Tools into Blender GUI

The third and the last stage of the project was implementing tools into the user interface. We aim to give users additional information about the animation of the neural network as well as to provide them some flexibilities to make managing the program easier. For this purpose, a custom panel called mitral panel is created in Blender that enables users to select mitral cells they wish to animate from the scene interactively. Instead of animating all mitral cells, users can select a small part of the whole network which they find more interesting.

Another feature that is added to the user interface is to setup a curve that shows the number of active mitral cells by a frame id. This feature can help users to select the time interval where they want to focus without spending time in other time intervals where no active mitrals exist, in other words where any mitral cell doesn't have electrical signal. In Figure 5, all somas are rendered at first by default. Users can select mitrals that they wish to render by

Mitral Panel and Activity Curve interactively selecting their somas. In addition to that, if they do not have enough information about the network, they can first play the animation and can see that which mitrals are active in which frame interval. The activity curve can also help users in deciding which time interval a number of mitral cells are more.

The report mainly contains technical background of the project. It can be found more visual elements in the video[2] published on the web.

References

[1] Migliore M., Cavarretta F., Hines ML., Shepherd GM. (2014). Distributed organisation of a brain microcircuit analysed by three-dimensional modelling: the olfactory bulb. *Frontiers in Computational Neuroscience*, 2014 Apr 29;8:50.

[2] SummerofHPC. (2015, August 27). 1505 Cem Benar, VisTool for olfactory bulb. *Retrieved from:* `https://www.youtube.com/watch?v=bQoNSMgKQ-k`

PRACE SoHPCVisTool for olfactory bulb
visualisation tool for olfactory bulb data

PRACE SoHPCSite
CINECA-Bologna, Italy

PRACE SoHPCAuthors
Cem Benar, [Ozyegin University,] Turkey

PRACE SoHPCMentor
Michele Migliore, Institute of Biophysics, CNR, Palermo, Italy

Cem Benar

PRACE SoHPCContact
Francesca Delli, Ponti, CINECA-Bologna
Phone: +39 051 6171506
E-mail: f.delliponti@cineca.it

PRACE SoHPCSoftware applied
Python, Blender 2.75

PRACE SoHPCMore Information
www.pa.ibf.cnr.it/personale/migliore/

PRACE SoHPCAcknowledgement
Special thanks to Prof. Michele Migliore, Luigi Calori, Francesca Delli Ponti, Silvano Imboden and Massimiliano Guarrasi

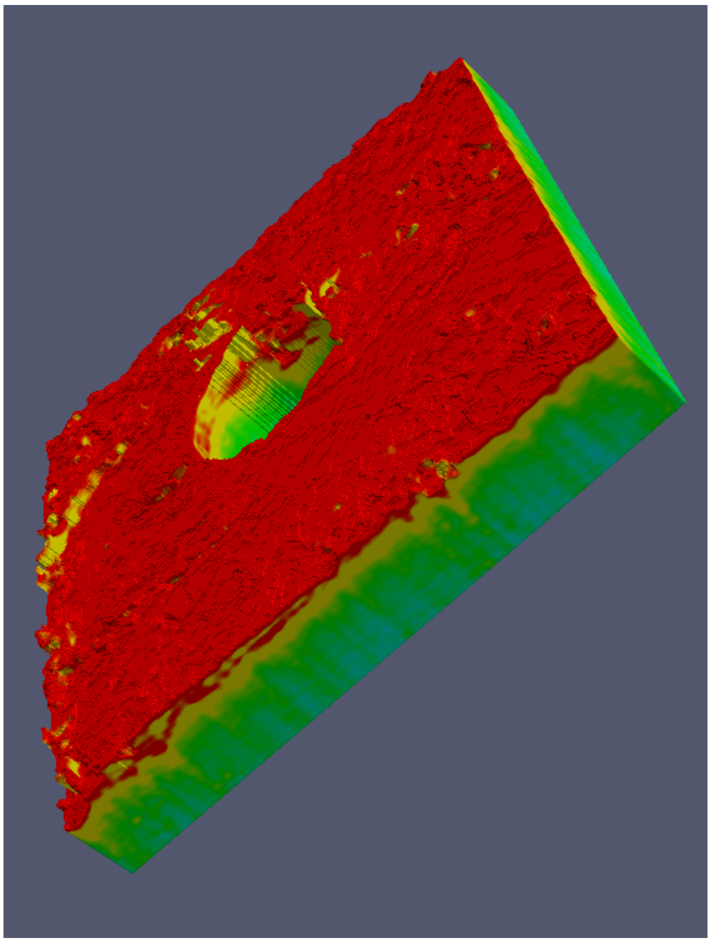
PRACE SoHPCProject ID
1505

Exploring Mars Surface Using 3D Visualisation of Post Processed MARSIS Data

Visualising Mars Pole

Leon Kocjančič

MARSIS is a radar carried by spacecraft Mars Express, and probing the interior of Mars to look for ice and water. Using radargrams that represent electromagnetic wave scattering, the 3D surface and subsurface layers of Mars' South Pole were visualised.



The reconstruction of the Mars south pole surface and subsurface layers using MARSIS data was divided into multiple consecutive steps. All of them are described below in the correct order. The most computationally demanding was the interpolation of the missing data points in a predefined matrix.

Where the data comes from

Mars Express is a spacecraft produced by the European Space Agency. Its journey to the Red Planet began on 2 June, 2003 with the launch of a Soyuz-Fregat Rocket. The satellite consists of many components for essential working and seven instruments to study all aspects of Mars.

One of the instruments is called MARSIS which is a low frequency, nadir-looking pulse limited radar sounder and altimeter. It operates at the altitudes of up to 800 km above the Martian surface for subsurface sounding and up to 1200 km for ionospheric sounding. The instrument is working in 1 MHz wide frequency bands centred at 1.8, 3.0, 4.0 and 5.0 MHz. It is transmitting a linear frequency modulated chirp signal with its dipole antenna. The returned signal is received on both the dipole and secondary monopole antenna. The secondary monopole antenna is oriented towards surface and it is used to reduce signal clutter from the ground.

Both signals from the monopole and the dipole antenna are converted to range offset video signals before they are processed by analogue into digital converter. Then the data is formatted by the on-board digital processor and passed to telecommunication module for transmission to Earth. The receiver electronics has local oscillator, which generates chirp signal, and dual channel receiver that down converts the received echoes. Each receiver channel has a bandpass filter, a mixer, an amplifier chain, low-pass filtering, and an analogue to digital converter. The receiver and digital electronics are housed together within a spacecraft. The transmitter electronics is housed in a separate box within satellite. The main transmitter and receiver antenna is a deployable dipole which consists of two 20 m elements. The extra gain towards the Martian surface is achieved through on-board processing.

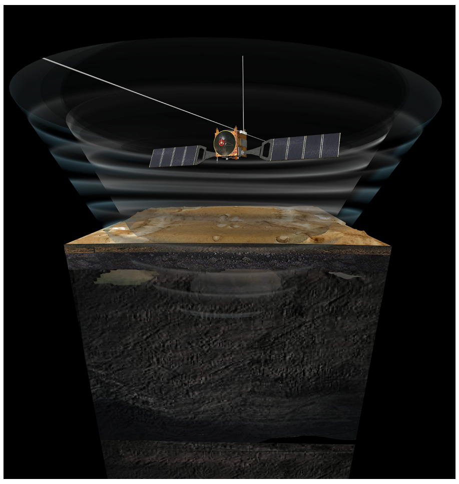


Figure 1: Image of Mars Express probing the Martian terrain.

The second monopole antenna is 7 m long and it is arranged in that way, so that its gain null is in the direction of Martial surface. Both antennas are of novel design and consist of a folding composite tube with two wires for signal transmission. The antennas were deployed by pyrotechnic release mechanism.

MARSIS performed subsurface sounding in the highly eccentric orbit selected for Mars Express, which corresponds to about 26 minutes of operation. This is equal to about 100 degrees of arc on the Martian surface for each satellite pass. During the mission extensive coverage at all latitudes was possible, because MARSIS supports dayside and night side operations. Performance of the ground penetrating radar is best during the night, which is when solar zenith angle is above 80 degrees. In this case ionosphere plasma frequency drops significantly and lower frequency modes of operation can be used, which can penetrate deeper in the ground. If lower frequencies are obstructed by ionosphere, higher frequency modes of operation are used and an on-board computer is constantly switching between them.

The primary objective of MARSIS is to map the distribution of water, both liquid and solid, in the upper portions of the crust of Mars. These discoveries will be very important for hydrological, geologic, climatic, and possible biologic research. The experiment has also three secondary objectives, which are subsurface geologic probing, surface characterisations, and ionosphere sounding.

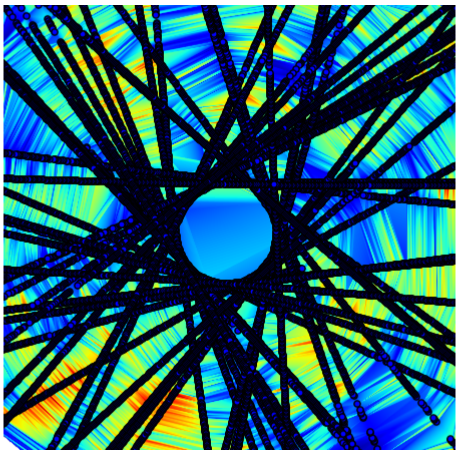


Figure 2: First 50 satellite tracks that were used for surface reconstruction.

Reading satellite data

Our first step incorporated reading the data that the radar module on Mars Express generated during its operation. Each satellite orbit around Mars is tracked and overall 38 parameters describe the various states of MARSIS module. All the parameters are stored in a well-organised binary file. Our collection of data consists of around 3000 satellite tracks that were passing over Mars South Pole.

The parameters that we were interested in were recordings of reflected radar pulses, as well as, longitude and latitude at which recordings were done. Radar is constantly scanning at two different frequencies and with three different filtered views at certain location. For our visualisation we have chosen first frequency, which is also the highest, in order to minimise distortion effects of ionosphere to the propagating signal. Additionally, we have used only the middle filtered view that contained visually the best samples of data.

During the importing routine of the data to our algorithm, we first checked if the gathered data lies south of the latitude of -75°. For all the coordinates that were in our interested field of view we applied the coordinate system transformation from spherical to Cartesian coordinate system. In this way we ensured simpler representation of all the data in the matrix. Additionally, it turned out that also interpolation was easier to implement in comparison to the case with polar coordinate system. At the end the gathered data covered a square surface of 1700 km by 1700 km over the Mars South Pole. Data was actually structured from radar echoes along satellite path, which were combined to radargram. However, the region closest to the pole is not covered since satellite orbits do not pass this area.

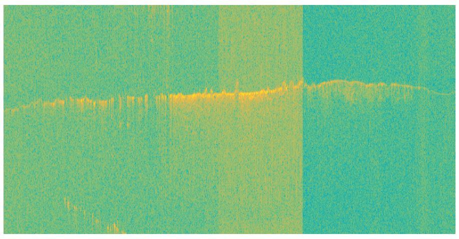
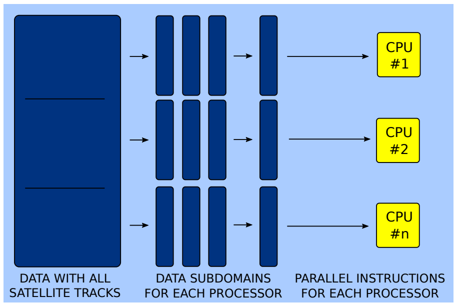


Figure 3: Example of a radargram that combines echoes along satellite path.

Right at the beginning we had to overcome first major technical difficulty. We predicted that our sequential code for data reading would run for more than 30 hours, which was more than our session allowed. At this stage we decided to parallelise the code to use all the available processing power at one computing node, which consisted of 20 processing cores. Since all our code was implemented in Python we decided to use multiprocessing library to parallelise reading routine. In this case we defined a pool of workers and each of them received only a part of the data to process. The order of calculation was not important that is why no synchronisation between processors was needed. The parallelised reading routine finished 10 times faster than sequential one.


CPU #1
CPU #2
CPU #n
DATA WITH ALL SATELLITE TRACKS
DATA SUBDOMAINS FOR EACH PROCESSOR
PARALLEL INSTRUCTIONS FOR EACH PROCESSOR
Figure 4: Schematics showing initial data decomposition and parallel instructions for multiple processors.

Interpolating missing data

Once all the data was imported we had to take care of all the missing values that were left in the matrix. The appropriate interpolation method was needed at this point. For testing purposes we used only 50 satellite tracks out of 3000 and immediately found out that the Scipy's barycentric interpolation method does not fulfil our needs.

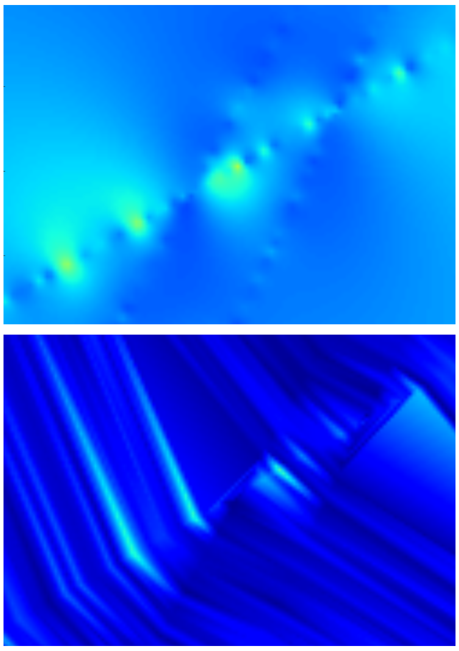
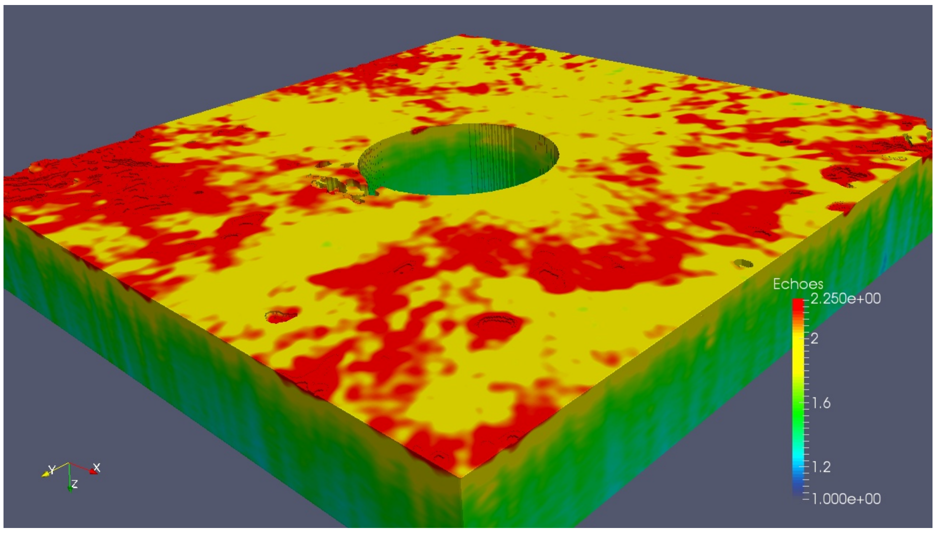


Figure 5: Comparison between barycentric interpolation method on top and RBF interpolation below.

One can see in the figure above that this interpolation method does not pro-



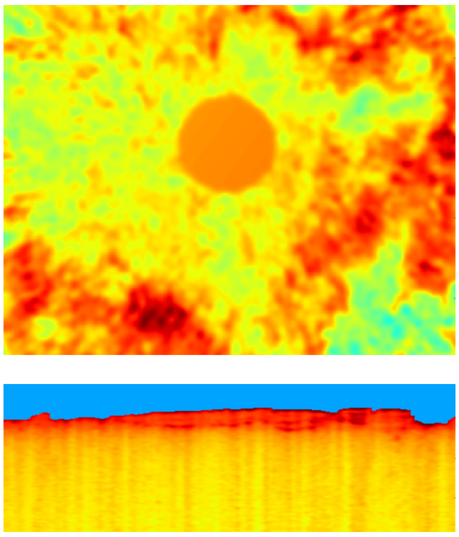


Figure on the left is xy-plane cut of the 3D model, in upper right corner is the result of 3D Gaussian filtering and below is the xz-plane cut where subsurface layers can be seen.

duce a smooth surface, which is expected to be seen after the process. Since the final results were not as expected we had to find better interpolation method. Fortunately, there exists more suitable method which is frequently used in topographical applications and is called radial basis function interpolation or RBF interpolation.

Using this method the interpolated result is actually the sum of the radial basis functions that are real-valued and depend only on the distance from the origin point. With the choice of appropriate function we can affect the smoothness of the end result. That is why the Gaussian function was used. During implementation of interpolating algorithm we found that the Numpy's RBF interpolation function was too slow so we have decided to use the algorithm in Alglib library.

The results were far better since this interpolation technique was also acting like a smoothing filter. With fine tuning of interpolation parameters that affect interpolation radius of each function we were able to achieve good results. For more time efficient calculation we used the same parallelisation technique as in case of data importing. This time the data was divided among z-planes and each process was interpolating its own plane. The interpolation lasted for 3 hours and the final result was 1.5 GB big matrix with surface data.

Data filtering

At this stage the interpolated data was not in the final form because the original data, gathered by the satellite, was quite noisy. It is quite an art to apply numerous digital filtering and still obtain nice realistic pictures of the recorded terrain. First filter that we implemented was cancelling negative values, which resulted at some specific points during the interpolation. When all values that are represented reflected power in data are positive we applied a filter for discarding data points with values below the value of average noise level. The filtering was done in the frequency domain using fast Fourier transform.

Our data points obtained from the satellite tracks were not evenly distributed and not perfectly matched along z-axis. This is why we had to introduce further filtering in order to smooth the image and better visualise the characteristics of the terrain. We constructed a three dimensional Gaussian filtering algorithm and applied it to the whole three dimensional data set. The filter had the range of 12 pixels in x and y direction and 5 pixels in z dimension. In this way we were able to produce good surface smoothing without losing the details of the subsurface layers in z-direction.

Final step

With all the data processed and filtered we could start to import it into Para View which is used for scientific visualisation. In this environment we were able to use some additional filters for cropping our data and removing central part of the region, which contained no real data. Then colouring of all the echo values contained in a matrix was done. We tried to select such colour scale that exposed also subsurface layers that were responsible for electromagnetic wave reflections. Finally fly-by movies and live cross sections were produced using ParaView's animation tools.

Conclusion

At the end all algorithms were successfully tuned in order to produce expected results. The library with used functions was created and will be used for further investigations of data produced by MARSIS module on Mars Express satellite.

PRACE SoHPCProject Title
Exploring Mars Surface – 3D Visualization of post processed MARSIS data

PRACE SoHPCSite
CINECA, Italy

PRACE SoHPCAuthors
Leon Kocjančič, University of Ljubljana, Slovenia

Leon Kocjančič photo

PRACE SoHPCMentors
Massimiliano Guarrasi, CINECA, Italy
Roberto Orosei, INAF, Italy

PRACE SoHPCContact
Massimiliano Guarrasi, CINECA
Phone: +39 051 6171 560
E-mail: m.guarrasi@cineca.it

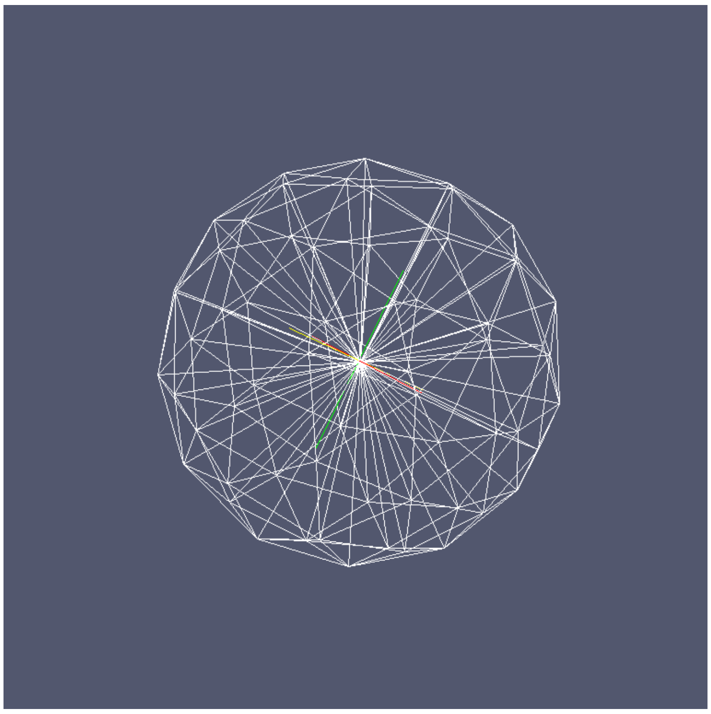
PRACE SoHPCSoftware applied
Python, ParaView

PRACE SoHPCMore Information
sci.esa.int/mars-express

PRACE SoHPCAcknowledgement
Author would like to thank both mentors and all supporting staff at CINECA.

PRACE SoHPCProject ID
1506

Investigating the parallel performance of the Fluctuating Finite Element Analysis tool



A closer look at FFEA

Jana Boltersdorf

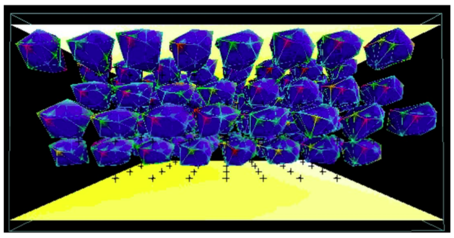
FFEA is a tool for simulating proteins and their environments which is currently limited to shared-memory systems. Understanding and improving its parallel performance will be crucial for FFEA's success.

FFEA has a lot of potential for simulating proteins at a scale currently not accessible by most simulation techniques. However, it is far too slow to be used for realistic, complex simulations even though an OpenMP parallelisation is already implemented. This project focused on gathering data about FFEA's parallel performance and how it might be improved in the future.

Introduction

Simulations have become an important aspect of scientific research, especially in fields where traditional experiments and observation are difficult to realise. But sometimes, even simulations are limited in scale: Large protein complexes are too computationally expensive for atomistic molecular dynamics but too small for conventional macroscopic simulation techniques to be applicable. FFEA aims to close this gap by using a Finite Element algorithm that has been generalised to include thermal fluctuations.

FFEA has the potential to open up new regimes of soft matter physics and molecular biology to computer simulation and become a standard tool in its field - but it needs to overcome its current technical limitations first! Right now, the code is limited to shared-memory systems which allows only small-scale simulations. My Summer of HPC project is to study and eventually improve the parallel performance of FFEA to enable the simulation of many proteins on a short timescale. This will require the usage of distributed-memory systems such as EPCC's ARCHER and a MPI parallelisation in addition to the current OpenMP approaches.



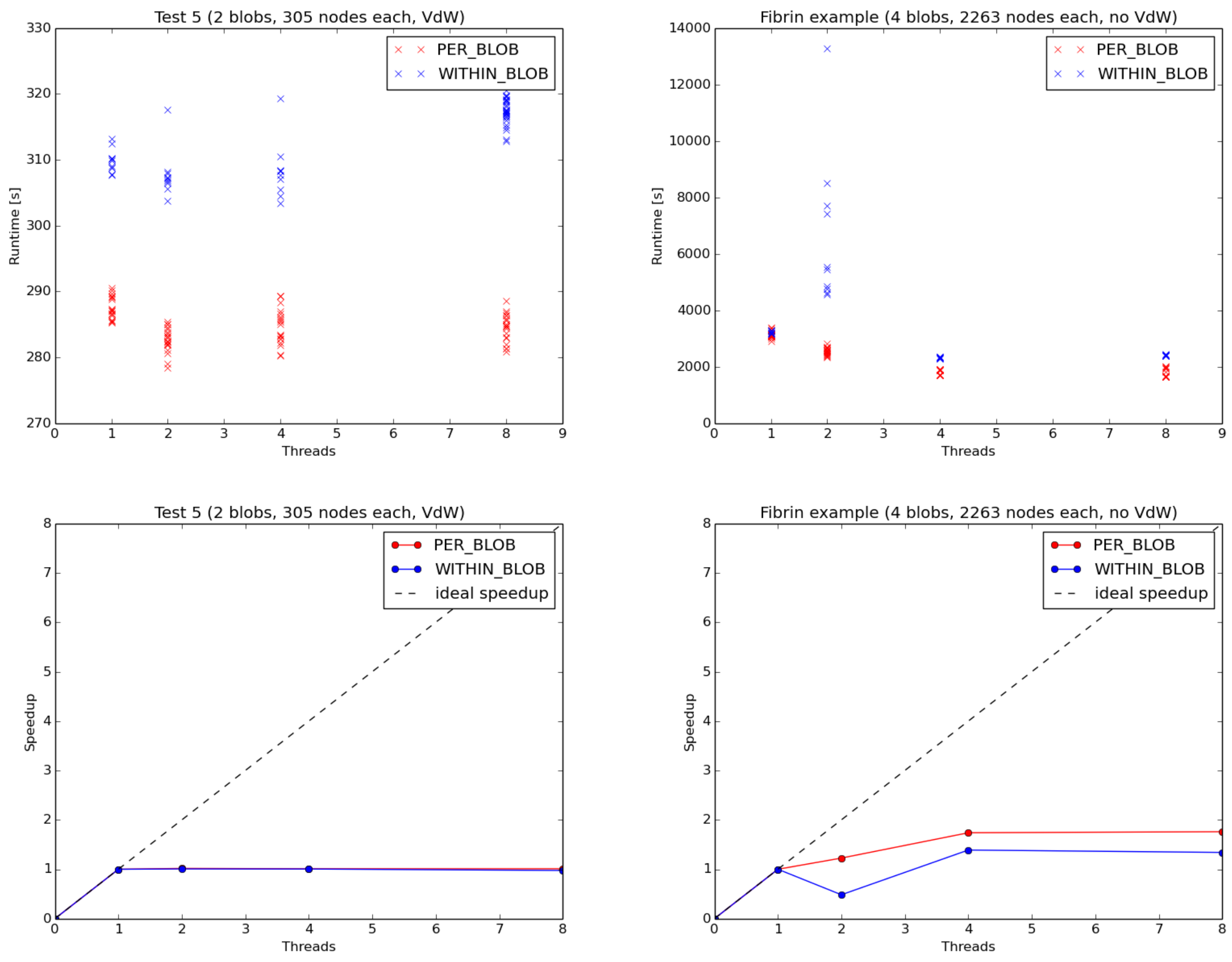
A simple simulation with many proteins blobs but few protein segment nodes per blob visualised by the old viewer.

The initial outline of my project sounded deceivingly straight-forward. Understand the basics of FFEA. Look for opportunities to parallelise. Use that information to design a simple MPI parallelisation. Implement that parallelisation. Test it. It was an ambitious goal even when assuming that everything sticks to the plan. Naturally, things didn't things didn't stick to the plan and many short delays accumulated. This is a very common thing in HPC as well as in science in general. The focus often shifts as you learn more about your project and its difficulties. In this way, it was a very valuable experience.
Nevertheless, my work here at EPCC helped to understand FFEA and its performance better. The gathered information will be used in the future as a part of a project spanning multiple years to improve FFEA's performance.

Methods

The first step when trying to improve a program written by someone else is getting familiar with it, trying to understand how it works and in general gathering as much information as possible. It's simple - you can't improve something you don't understand.

I started by running the test script provided by the Computational Biophysics group of the University of Leeds, then read into the actual source code. Especially the classes `World.cpp` – representing the whole simulated cellular environment – and `Blob.cpp` – representing one protein in this environment – were of interest. These tasks



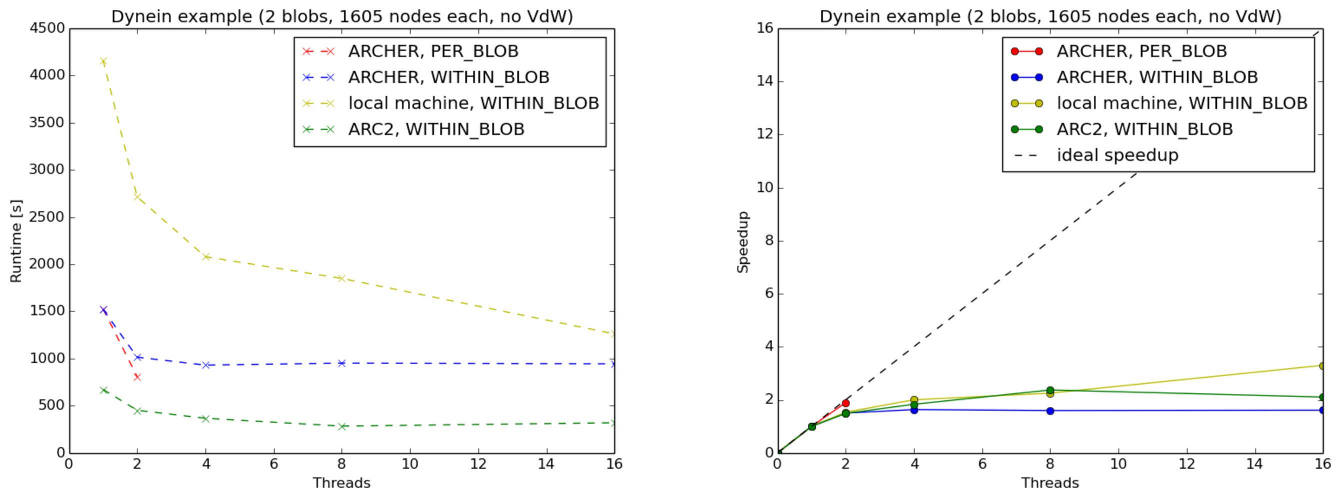
Left: Runtimes and speedup of the initial test with two protein blobs, X protein segment nodes each and simulated Van der Waals interactions. The simulation was far too simplistic for the parallelisation to have a significant effect.

Right: Run times and speedup of the simplified Fibrin test with four protein blobs, 2253 protein segment nodes each and no simulated Van der Waals interactions. The effect of the parallelisation is now visible but even for a small number of threads still far below the ideal speedup.

provided me with a basic understanding how FFEA represents the environment it simulates and what results can be expected for very simple simulations.
But since FFEA already had an OpenMP parallelisation, I was also interested in its current parallel performance. There were two different parallelisation strategies available: within-blob parallelisation and per-blob parallelisation.
Within-blob parallelisation – which uses all available threads for each instance of `Blob.cpp` – is the default parallelisation. It can be switched to per-blob parallelisation – assigning one thread to each instance of `Blob.cpp` to handle multiple instances simultaneously – using CMake. As there were several problems with CMake that delayed my progress, I resorted to editing the Makefile manually. While this wasn't the optimal way of changing the parallelisation strategy, it allowed me to re-compile the code using the per-blob parallelisation.

To measure the performance with both parallelisation strategies, I ran many tests on my local machine and compiled the measured runtime. At this point, I noticed that the simple simulations used had a fundamental drawback: They were too short for the parallelisation to have any effect.
A bigger, more complex test was needed and provided by the Computational Biophysics group... Just to have another problem to emerge: It was far too big and complex. After three days, the simulation still hadn't finished and remember... I had to run it several times to get the required data about the parallel performance. The solution was simple: Edit the input file and decrease the simulation size. However, it took some time to find a good balance between runtime and complexity of the simulation. The restart option – re-using data from previous calculations – proved to be particularly challenging as Van der Waals interactions could only be calculated with this feature. On the other hand, getting reliable runtimes required the simulations to do the whole simulation rather than re-using data. To resolve this dilemma, I decided against the Van der Waals interactions at this stage. It made the simulation less realistic but easier to handle compared to the necessity to manually delete trajectory data after each successful run. I now was able to compare the runtimes and calculate the speedup of both parallelisation strategies.

The next step was supposed to be



Runtimes and speedup of the Dynein test with two protein blobs, 1605 protein segment nodes each and no simulated Van der Waals interactions. The local machine and ARC2 runtimes were measured by the Computational Biophysics group in Leeds. Note that there are only two values for the per-blob parallelisation strategy as the other runs aborted because of inverted elements.

some more sophisticated profiling than just comparing the overall effect of the different parallelisation strategies on the runtime. This also made the move from a local machine to ARCHER, the UK's national supercomputing service, necessary. It didn't happen as smoothly as I'd have wished but in the end, I was able to gather more data about FFEA's parallel performance using another example and compare it to data previously collected by me as well as the Computational Biophysics group.
Problems with running SCALASCA – a popular profiling tool – on ARCHER as well as the limited time-frame of the project prevented me from actually starting to profile the code though.

Results

The collected data suggests that both parallelisation strategies currently used in FFEA only use a fraction of the potential parallel performance. Due to serial sections of the code, very few programs scale perfectly but this only becomes obvious for large numbers of computation cores. In the case of FFEA, the parallel performance is especially limited: In many cases, the maximum speedup is achieved with just four cores. That's what an average computer nowadays has.
The differences between the parallelisation strategies are small, both show the same tendencies regarding their behaviour. This behaviour has also been observed on all systems used for calculations (two different local machines, ARCHER, ARC2) by both the Computational Biopysics group and the EPCC.

For environments with just one complex simulated protein, the within-blob parallelisation shows better results. On the other hand, for almost all systems with more than one simulated protein, the per-blob parallelisation strategy performs better.
In the last plot, you might have noticed that there are only two values for the per-blob parallelisation. The other runs were aborted due to an inverted element error. A closer investigation of the cause showed that they were caused by random thermal fluctuation and could have happened to any other run as well.

Discussion

Although the members of the Computational Biophysics group considered FFEA's parallel performance for simulations without Van der Waals interactions okay, there is still a lot of potential to be unlocked. Right now, the code doesn't even come close to the limits of shared-memory systems. This should be improved before heading to distributed-memory systems.
A necessary step will be the profiling of FFEA with a tool such as SCALSCA or CrayPAT. Hopefully, this will provide more insights into the reasons of the bad parallel performance I observed... And therefore also hint at parts that could be improved.

A hybrid OpenMP/MPI parallelisation will be needed for FFEA to unlock its full potential in the future.
In the short term, a MPI parallelisation modelled after the current per-blob parallelisation seems promising as it requires only little communication between the different processes. A long-term goal will be the parallel calculation of the Van der Waals interactions. Right now, they cause a severe slow-down but they won't be easy to parallelise. To establish an excellent parallel performance in the future, this difficulty has to be overcome though.

Acknowledgements

Albert Solernou, Ben Hanson and Sarah Harris from the Computational Biophysics Group at the University of Leeds created FFEA and provided me with many explanations about it as well as examples to measure the code's performance.

PRACE SoHPCProject Title
Implementing MPI parallelisation in the Fluctuating Finite Element Analysis (FFEA) tool

PRACE SoHPCSite
EPCC, UK

PRACE SoHPCAuthors
Jana Boltersdorf, FH Aachen, Germany

PRACE SoHPCMentors
Neelofer Banglawala, EPCC, UK
Toni Collis, EPCC, UK

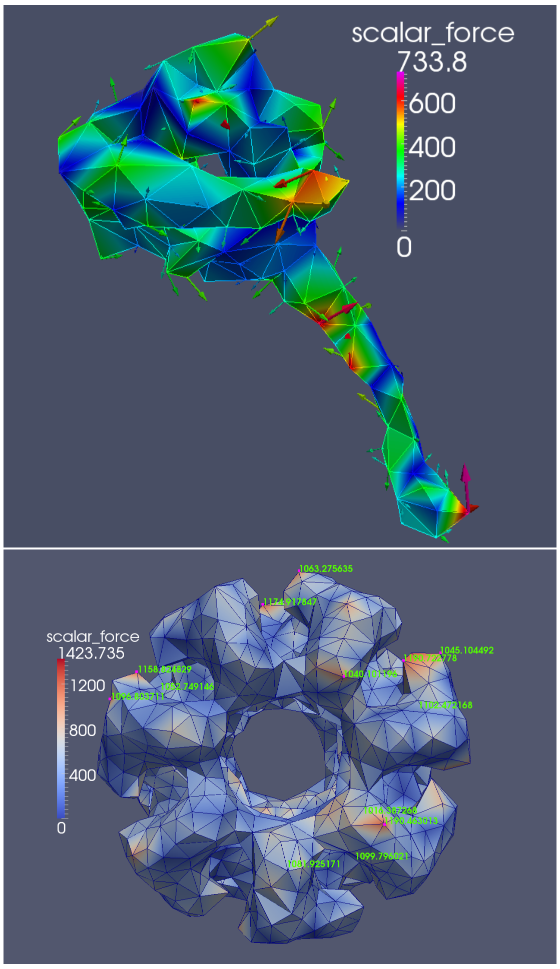
Jana Boltersdorf

PRACE SoHPCContact
Jana Boltersdorf
Phone: +49 157 88062834
E-mail: j.l.boltersdorf@gmail.com

PRACE SoHPCAcknowledgement
Albert Solernou, Ben Hanson & Sarah Harris, Computational Biophysics Group, University of Leeds

PRACE SoHPCProject ID
1507

Developing the user interface for the Fluctuating Finite Element Analysis (FFEA) tool



Visualisation of proteins

Ondřej Vysocký

Fluctuating Finite Element Analysis is a method for simulating proteins. ParaView is a visualisation tool with which it is easy to observe and manipulate the results of the simulation.

The Fluctuating Finite Element Analysis tool for soft macromolecules simulation needs a fast and reliable visualisation tool, which allows visualising simulations results from many points of view.

The widely used visualisation tool ParaView provides scientists with an easy way to visualise data. It also allows scientists to interact with data.

Introduction

The Computational Biophysics group at the University of Leeds would like to model the dynamics of proteins. Usually protein simulations, such as fibrin aggregation, are typically at the atomic level, making them so detailed that the computational expense of the calculations and simulations becomes unfeasible. As a solution to this problem, the Computational Biophysics Group at Leeds has developed a coarse-grained model for proteins using finite element analysis that includes thermal fluctuations. This is known as the FFEA (Fluctuating Finite Element Analysis) tool.

This report explains the work carried out to improve the visualisation interface of FFEA to enable the handling of the large data sets that are typically produced by the FFEA simulations.

Methods

The initial idea of the project was to enhance the existing visualisation interface, a Python viewer is written in TKinter, which is Python's de-facto standard GUI. This viewer could only visualise basic features (such as protein structure or surface) and its performance was poor. It also could not handle visualising large scientific datasets.

PID	**USER**	**PRI**
27543	v1ovysoc	24
NI	**VIRT**	**RES**
4	6583m	6.1g
SHR	**S**	**CPU%**
3968	R	75.8
MEM%	**TIME+**	**Comand**
78.7	27:48.24	python

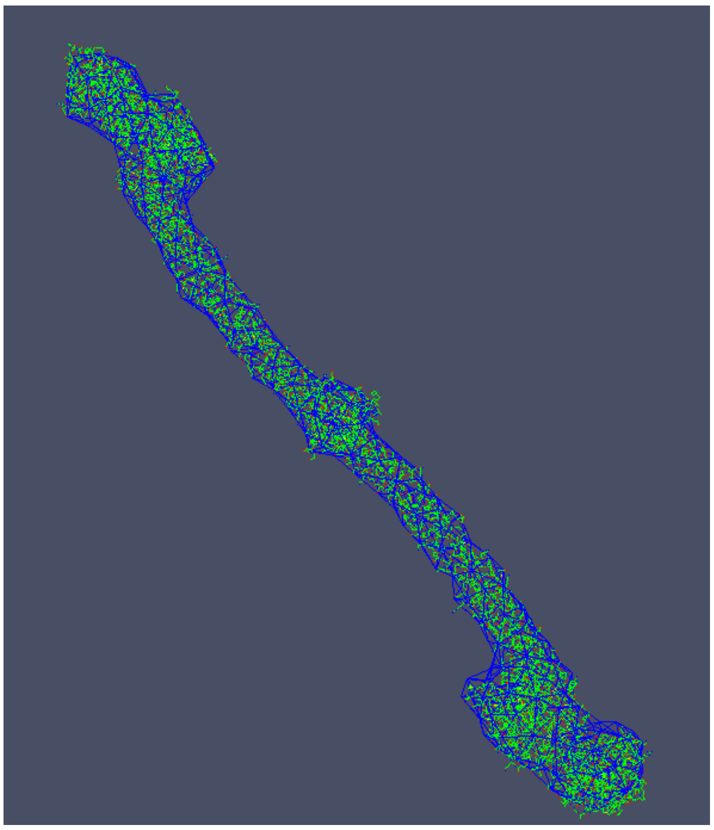
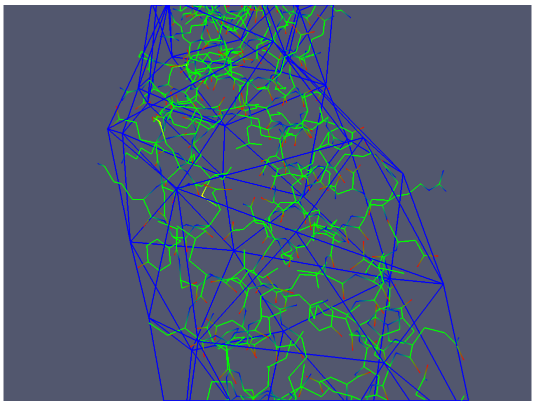
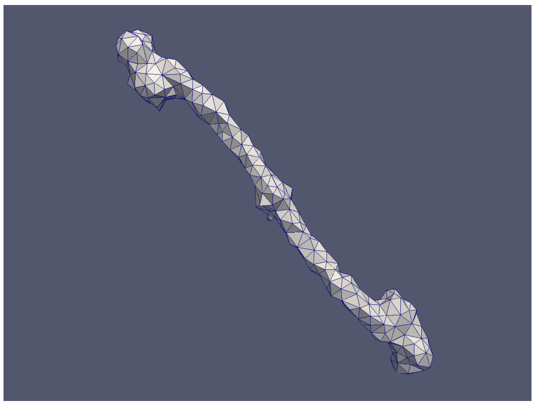
Table 1: htop analyse of previous visualisation tool

In table 1 (linux utility htop output) we can see the CPU and memory consumption of the old viewer when visualising simple simulation data. Larger datasets are almost impossible to visualise with this viewer.

The best idea to solve the problem of slow visualisation was to replace the viewer with a widely used visualisation tool. We chose the well-known ParaView, which is an open-source, multi-platform data analysis and visualisation application. To visualise the simulation data on ParaView it needed to be converted to a format recognised by ParaView. I wrote a python script to transform the simulation output data to the VTK (The Visualisation Toolkit) file format, which is a popular file format supported by Paraview.

A VTK file contains positions of each point of the structure, connections of the points to the cells (there are many types of cell structures). Also each point and each cell may represent a specific value, which can be visualised with different colours. Values might be scalars and also vectors, which allow us to visualise not only intensity but also the direction of the measured variable.

To be able to convert the simulation output data to a VTK file format it was necessary to understand the con-




PDB-VTK mapping in images. Right column figures: top figure shows the atomic structure of the protein from a PDB file, bottom figure shows the VTK representation of the same protein. In the left picture are PDB and VTK versions of the protein superimposed.

tent of the simulation's input and output files as well as the VTK file format. Application of advanced visualisation features required detailed understanding of ParaView's built-in functions.

Results

The main result of this work has been to make it easier to visualise simulation output data, especially large data sets, using a powerful and widely-used visualisation tool.

Result of the simulation is not only one state of the mesh, but series of states, each for one time-step. For scientists it is important to have an opportunity to watch the evolution of the simulation in consequent steps.

Now we are able to view the protein trajectories as ParaView animations, which we may manipulate to, for example, change the camera position. It is also possible make a video out of such animations. ParaView can create a video directly or produce a sequence of images, which we can then transform into video using software such as FFmpeg (widely used, freely available software).

New visualisation features are now possible using ParaView's built-in functions. For example, we may set a threshold for a specific variable and select points or cells we are interested in within that threshold.

Proteins we want to simulate are often known at the atomic level and stored in Protein Data Bank (PDB) files. It means that we have three-dimensional structures of molecules of protein in one file and also mesh surface and basic structure in another file (which is the product of the FFEA tool). In ParaView we may visualise both of them and see atomic structure superimposed to the mesh representation. With the Python convert script it is also possible to see atomic structure superimposed on the mesh FFEA representation of the same protein, which give us very important view at the simulation results with large amount of information in one image. This was not possible in the old viewer.

All features settings are documented in a detailed tutorial.

Conclusion

By using ParaView, we may now only visualise all the features that were present in the old Python visualisation tool, we can additionally visualise new features (most of all PDB-VTK mapping), making it easier to understand the simulation output data. It is now also possible to make a video of the protein trajectories, making it easier to present simulation results to the general public.

PRACE SoHPCProject Title
Developing the user interface for the Fluctuating Finite Element Analysis (FFEA) tool

PRACE SoHPCSite
Edinburgh Parallel Computing Centre at The University of Edinburgh, Scotland

PRACE SoHPCAuthors
Ondřej Vysocký,
Brno University of Technology,
Czech Republic

Ondřej Vysocký

PRACE SoHPCMentors
Neelofer Banglawala, EPCC, Scotland
Toni Collis, EPCC, Scotland

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

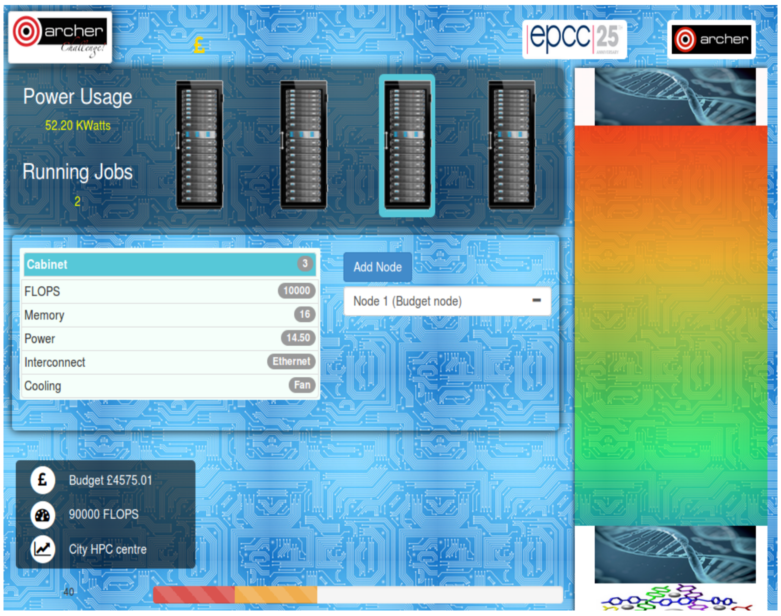
PRACE SoHPCSoftware applied
ParaView, VTK, python, FFmpeg

PRACE SoHPCProjectID
1508

ARCHER Challenge

Anna Chantzoplaki

The aim of the application is to promote High Performance Computing and its importance on solving computational intensive problems to the public outreach. The users will be able to design their own Supercomputer cluster, by picking and assembling predetermined component parts, while working within a fixed virtual budget.



Cabinet	3
FLOPS	10000
Memory	16
Power	14.50
Interconnect	Ethernet
Cooling	Fan

ARCHER is the name of the UK National Supercomputing Service, housed at the University of Edinburgh's Advanced Computing Facility. That inspired the initial idea of creating an application that could simulate a supercomputer's behaviour and which could be freely available to download.

The aim of the application is to promote High Performance Computing at the public outreach and its importance on solving computationally intensive problems. The final product will be a mobile/web application, specifically designed to be used at science fairs, schools, and other outreach events and it could be also used as part of PRACE outreach events.

The technologies that were used are the HTML language, the CSS for the styling of the page and the Javascript for the dynamic handling of the events of the page.

Methods

More specifically concerning the technologies that were used, the application is developed based on the Bootstrap framework, one of the most popular HTML, CSS and Javascript frameworks for developing responsive applications. The term responsive refers to the scaling of the application on different screen sizes. The Bootstrap framework scales the application easily and efficiently.

In addition to that, the Jquery library was used. JQuery is a javascript library that allows writing simple code, to have easy access on the elements of the page in order to modify them dynamically; it supports extensive chaining, and has many more features which make it powerful.

Moreover, the project focused on the mobile-first implementation as a default layer to build on, according to the fundamental concepts of modern web designing.

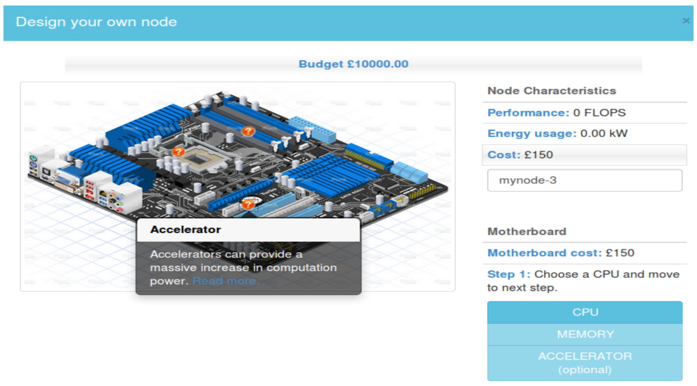
Terminology

In this point it would be useful to describe some very basic terminology that is used in High Performance Computing (HPC) systems, since it is also being used by the application.

A **node** is like a simple computer. It consists of a motherboard, CPUs and accelerators (GPUs). The nodes of a supercomputer are placed into **cabinets**, which also contain the networking and cooling systems. The term **job** is referring to HPC programs that are submitted to nodes and need huge computational power in order to be executed. These jobs usually run on huge data sets in order to solve large problems in science, engineering, or business. Some examples include simulation of earth's climate, extreme weather forecasting, molecular dynamics simulation and many more. Finally, performance of a supercomputer is measured in floating point operations per second or just **FLOPS**.

Results

The final product is a game for phones, tablets and desktops. The game has four levels (Local, City, Regional, National) each referring to an HPC centre. The player starts at the Local level and with a predetermined amount of money. A genius guy, the Technical Advisor, will pop up from time to time to give advice to users on which step to follow next. In the pursuit of simplicity, there are four available cabinets and each cabinet fits up to three nodes. There is also, a stack with queued HPC jobs. The target is to run as many jobs as possible. The more jobs the player runs, the more money the player earns. An effect of money coming out of cabinets is displayed on Figure 2a.


Design your own node

Budget £10000.00

Accelerator

Accelerators can provide a massive increase in computation power. Read more.

Node Characteristics

Performance: 0 FLOPS

Energy usage: 0.00 kW

Cost: £150

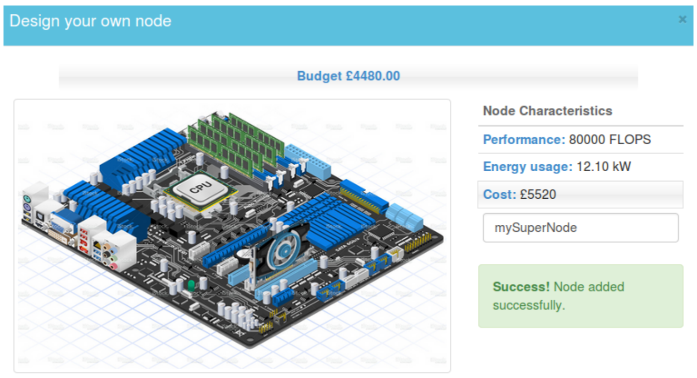
mynode-3

Motherboard

Motherboard cost: £150

Step 1: Choose a CPU and move to next step.

(a) Custom node wizard, with interactive motherboard.


Design your own node

Budget £4480.00

Node Characteristics

Performance: 80000 FLOPS

Energy usage: 12.10 kW

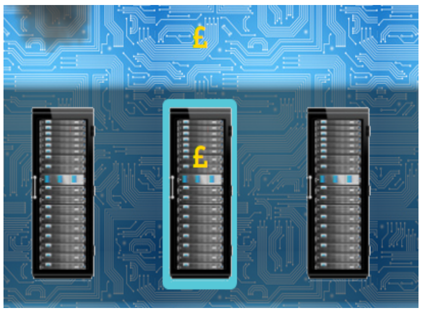
Cost: £5520

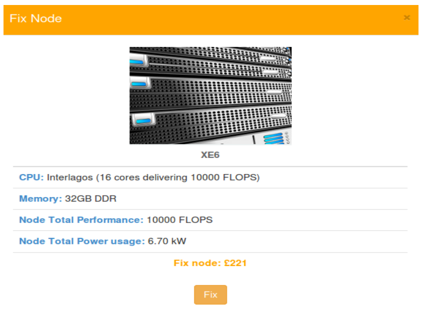
mySuperNode

Success! Node added successfully.

(b) On final step the components are placed on top of motherboard.

Figure 1: Node designer wizard.




Fix Node

XE6

CPU: Interlagos (16 cores delivering 10000 FLOPS)

Memory: 32GB DDR

Node Total Performance: 10000 FLOPS

Node Total Power usage: 6.70 kW

Fix node: £221

(a) Left node: failure, need to be fixed. Middle node: a finished job yeilds profit.

(b) Fix the node by clicking on the button.

Figure 2: Different aspects of the "ARCHER Challengge" app.

The player will be able to select between preconfigured nodes, or make a custom one, as it is shown on Figures 1a, 1b. In the case of a custom node, a wizard will guide the player through the process of picking components and assembling a node. There will be also the option to upgrade a node or even delete it.

Since node failures on a cluster are more than common and expected, the game incorporates it in a higher level (City level). Smoke will start coming out of cabinets, Figure 2a, and the Technical adviser will prompt the player to select the cabinet and fix it, as shown on Figure 2b. By fixing an amount of money will be subtracted from the existing budget. As the player passes on higher levels, the pace of the game becomes faster. More nodes fail and more jobs run into the queue. Be careful, though. If the stack overflows, the game is over.

Discussion & Conclusion

The project aims to give people an insight into the different components that go into a supercomputer. Also, it is a simple way for people to get familiar with high performance computing and to understand the impact it has on real-world implementations.

It is worth mentioning that the first version of the game was already presented to a school in Edinburgh in order to get a feedback on how the application feels. The reactions were more than positive and the users showed great interest in learning more about HPC. That was the first of the many outreach events that the supercomputing application will be demonstrated and used by the public. Finally, the ARCHER website will host the application and people will be able to play with it on-line or download it. The expectations are that the "ARCHER Challenge" will become viral.

PRACE SoHPCProject Title
An "HPC Cluster Challenge" app for public outreach

PRACE SoHPCSite
Edinburgh Parallel Computing Centre, University of Edinburgh, UK

PRACE SoHPCAuthors
Anna Chantzoplaki,
University of Macedonia, Greece

Anna Chantzoplaki

PRACE SoHPCMentors
Nick Brown, EPCC, UK
Amy Krause, EPCC, UK

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

PRACE SoHPCSoftware applied
HTML, CSS, Javascript

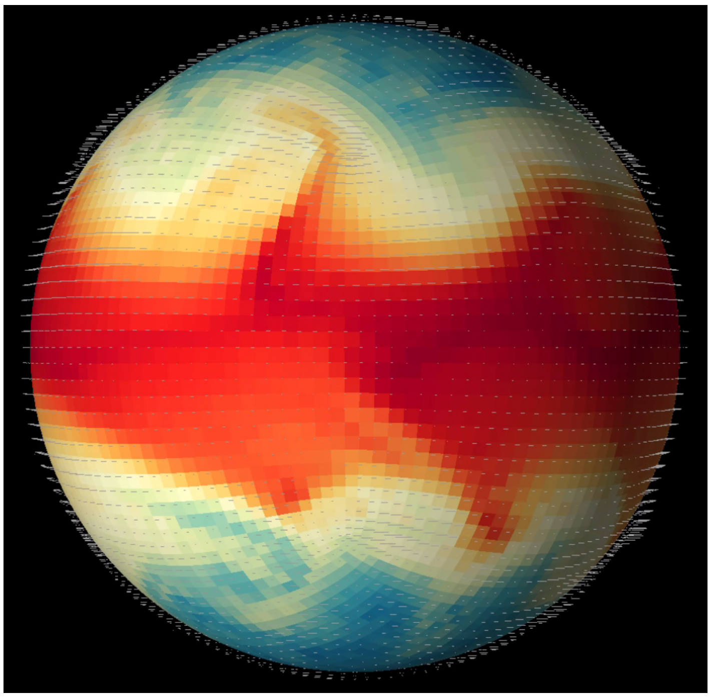
PRACE SoHPCAcknowledgement
I would like to thank my mentors Nick and Amy, and my colleague Emmanouil Farsarakis for all their help during this summer.

PRACE SoHPCProject ID
1509

Various Weather Regimes during
Continuous Climate Simulations

Climate is unstable

Ondřej Ticháček



Climate is unstable during a continuous change of its boundary conditions. The change of one parameter can reveal several possible climate regimes.

Climate and weather systems are one of the most important processes to study, however given their complexity also one of the most difficult to understand. One cannot possibly figure out all the relations and dependencies between all of the numerous factors involved. There is no universal formula to tell us what the weather will be tomorrow. Furthermore, from our perspective, weather is a *stochastic process*. In other words, even if we knew such formula, we could not be sure about our prediction.

While weather is a stochastic process, in a large-scale, long-time averaged sense, climate is primarily a deterministic function of its *boundary conditions*, and responds to changes in those boundary conditions in sometimes dramatic ways.

The goal of this project was not to create a better or more advanced climate model but rather to show interesting dependencies between the model output, i.e. the global characteristics of the **climate**, and certain **input parameters** all-together defining the boundary conditions. These parameters include the planet radius, gravity, rotation rate and others.

While the changes of these variables are certainly impractical, they should serve to show how the climate we have is only one from a wide spectrum of possible *climate regimes*.

The simulation is driven by the dynamical core of a general circulation model (GCM) of the atmosphere, specifically the MITgcm (mitgcm.org). This is a large open source project being in constant development for almost two decades. It allows the user to choose from various predefined configurations or implement of their own. For our project, we used the *Held–Suarez configuration*,[1] which is now a classic benchmark test for climate models. This setup ensures that the simulation can be integrated forward many years without the need to use too extensive resources. In principle, the use of a more advanced (and more computationally demanding) model is just a matter of configuration.

The ultimate goal was to develop a package for the MITgcm newly implementing the capability to change any physical configuration parameters **continuously** during the simulation, thus allowing new class of simulations. Additional work was required in setting up the model on the supercomputer fionn at Irish Centre for High-End Computing (ICHEC.ie) and in implementing means of visualisation of the model output.

The model

All general circulation models models attempt to solve representations of Newton's equations of motion (e.g., the Navier–Stokes equations) everywhere in a fluid that is varying continuously in space and time. The continuous fluid must be discretized in both space and time, and the differential equations replaced with finite-difference algebraic approximations.

Such approximations inevitably introduce errors (e.g., round-off errors), but various techniques have been developed to minimise and control such errors. The key to minimising the errors is a good choice of the *discretisation*. Usually, a general rule applies, that smaller grid-size requires a smaller time-step for numerical stability. Therefore, to ensure numerical stability a fine balance between high enough resolution and low enough computational requirements must be found. With low resolution, i.e. if two nearest points are too far apart (either in time or space domain), the error of the approximation is likely to be high, while high resolution may be too computationally demanding.

It should be noted, that actual resolution is very relative. For example, the distance between two consecutive time points, commonly called the *time step*, in a simulation of the atmosphere can be as high as couple of minutes, whereas in an atomistic simulation of a molecule, the time step is in order of femtoseconds, that is 10^{-15} sec.

In the field of climate and weather modelling, the discretisation in the space domain is the cause of more issues than the time discretisation and is therefore more interesting to study. The most intuitive way of dividing a space domain is, either 2D or 3D, rectangular grid. This grid has many benefits originating in the same-size steps between grid points and in the right angle between them. However, the rectangular discretisation is not possible with a spherical object, such as the Earth.

Of course, a possible solution to this problem would be to project the Earth on a rectangular plane – as it is commonly done in maps – the so-called mercator projection, but this has other more serious problems. For example, in mercator projection, Greenland and Africa are approximately the same size, whereas Africa is in reality 14 times larger. The real solution is therefore not to project the Earth on a rectangular grid, but otherwise, to project a grid onto the Earth, even if the grid is not regular nor rectangular.

There are several options for the sphere discretisation, the MITgcm implements two of them: the **spectral model** using the regular latitude-longitude (lat-lon) grid and the more recent **cubed sphere** (CS) grid model. Each of the approaches has it's benefits and drawbacks, but in terms of high performance computing, the cubed sphere model is usually deemed more advanced. The difference between cubed sphere and lat-lon grid is illustrated in the figure 1.

The first disadvantage of the lat-lon grid is a numerical one. The longitude-wise spacing between the grid points at the poles is much smaller than at the equator, which may cause, as was discussed earlier, numerical stability issues or extensive computational requirements.

The second problem arises in terms of high performance computing. Sometimes it may be impossible to perform the simulation on a single computer; this is typically the case when the requirements for the spatial or time resolution are high or when the model is complicated and contains many dependent equations. The only solution for this problem is *parallelism*, i.e. dividing of the task into smaller sub-tasks, and solving these sub-tasks on multiple processors or computers at the same time. This is not possible in the time domain as every single simulation time step is dependent on the time steps before.

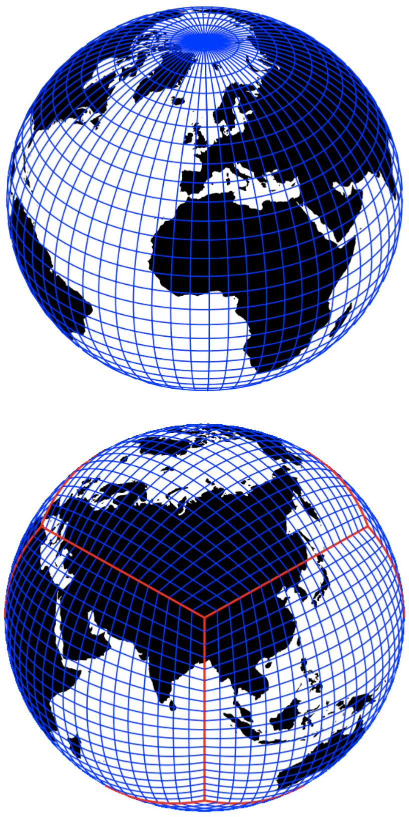


Figure 1: The regular latitude-longitude grid and the cubed sphere grid. Credit: Paul Ullrich

In the space domain, on the other hand, only few neighbouring grid points are effectively dependent at every single time iteration[1] and therefore, the spatial grid can be divided into parts, each of them assigned to a different task. Of course, the grid points at the borders are still dependent and the task must communicate with each other, which is mostly the stumbling-block of each parallel application. And here, finally, comes the origin of the second problem with the spectral model – communication.

The spectral model represents all variables with a "spectrum" of waves, and does all linear operations in wave-space, since derivatives in wave-space are exact. However, non-linear operations are done in grid-space. So spectral models must transform from wave to grid space and back again every time-step, and such transforms require data transposes which invariably involve a so-called *all-to-all communication*, which is the most computationally expensive type of communication. The cubed-sphere, on the other hand, is a purely grid-point model, so there is no need to transform to wave-space, and so only "nearest-neighbour" MPI communication is needed.

The drawback is that the cubed-sphere model sacrifices some accuracy in its derivatives, since it uses grid-point finite-differencing instead of the exact differentiation that is possible within the spectral model.

My work

Setting up the simulation

The MITgcm is a huge software package. Apart from the core, it includes several packages implementing different I/O capabilities, spatial domain discretisations, external forcing configurations, and others. My job for was to get familiar with the model and its capabilities and select a basic configuration for future simulations. Based on the reasons discussed above, I selected the cubed sphere discretisation, 64×64 grid points per face. The cubed sphere configuration requires so-called grid files, describing the position and orientation of each point. These files are not included and must be generated using a provided MATLAB program.

The MITgcm employs both MPI and OpenMP mapping processes and threads to so-called *tiles*. Given that our goal was to run the MITgcm on a single fionn node, I configured spatial domain decomposition to tiles of a size 32×32, 4 tiles per face, 6 faces in cube resulting in 24 MPI processes matching the 24 physical cores of a fionn node. The non-default 64×64 CS configuration required to tweak domain decomposition overlap, time-stepping and other numerical parameters to ensure numerical stability.

Since the atmosphere is relatively well "stratified", however (i.e., hydrostatic balance means it is much more difficult to displace air vertically than horizontally), much higher resolution is needed in the horizontal directions than in the vertical. For the horizontal resolution, 20 equidistant levels were selected.

The mnc package

Appart from the standard fortran file I/O, the MITgcm can use the NetCDF file format, a self-describing file for-

[1]Through the whole simulation time span, all grid points are essentially dependent.

mat commonly used for multidimensional array oriented scientific data. This functionality is provided by a package named `mnc`. The outputs of the model, e.g. the fields of temperature, wind velocity, etc., are saved into a common file, while other statistics such as time-averaged fields, variables describing the grid, debugging and monitoring outputs and others are each written in separate files. There is no problem with the `mnc` package, when running the model on a desktop computer, however on supercomputers, which are often using very different file systems, some issues may arise.

Similarly to the yet mentioned problem with communication between different sub-processes of an application, the problem with the file system on supercomputer is file I/O. In principle, the computational node, i.e. the "processor" of a supercomputer may be very far away from the storage, i.e. the "hard disk". The link between them may be very fast for long sequential transfers of large files, but randomly accessing files may be slowed significantly by the latency. Therefore, the computational nodes usually have a fast small storage placed physically at the node's location and the applications are required to use this so-called *scratch* storage during the computation, and copy the results to the main storage after the whole computation finishes.

An application can sometimes benefit even more from using the shared memory (`/dev/shm`) storage, which is basically a storage inside RAM. The speed of accessing RAM is of course orders of magnitude faster than conventional hard drive, however, their sizes are also orders of magnitude apart. On the fionn supercomputer, RAM of each computational node is 64 GB. This is not enough for storing a long high resolution simulation of MITgcm, but still can be a beneficial.

As a part of my project, I implemented two methods of dealing with this problem. The first solution was to divide the simulation into sequential parts, each storing the results in the shared memory and copying them to the main storage between the parts. The second solution required a modification to the `mnc` package – to allow each file either to be temporarily stored in the shared memory and copied after the end of the simulation or to be continuously updated directly at the main storage though the whole simulation. An evaluation of write frequencies and file sizes of all output files was required and based on this properties, the only the file with snapshots of physical fields and the file with time averaged fields are updated directly at the main storage. With my configuration, these files accounted for more than 90 % of storage while their update frequency was significantly lower than the other files'. As it is strongly dependent of the simulation configuration, the save location for each of the files can be modified in the newly implemented feature of the `mnc` package.

Slow evolution of parameters

The MITgcm program can be summarised by following pseudocode.

```
load_params()
params_dependencies()
for t = 1 to endtime do
    evaluate(t)
    write_output(t)
endfor
```

Once parameters are set, they can not be played with. As a part of my project, I wrote a MITgcm package called `slowevo` which enables evolution of parameters during the simulation by ensuring, that all the dependent parameters and variables are still consistent. Currently, only few changeable parameters are implemented at the interface of the package, however in the core, all of the model parameters are tracked for changes and their dependencies updated. Other parameters can be added to the interface very simply if needed. The implemented physical parameters include the rotation rate of the planet, the gravity and the planet radius as these are the parameters in the Held–Suarez configuration. To illustrate the package utilisation in more real-life application, we can imagine a simulation of the Earth's climate with a externally changing concentration of carbon dioxide as a parameter.

The package respects the MITgcm conventions for enabling (including in compilation), loading (run-time selection of package usage), package internal parameters input, run-time monitoring and error reporting.

Post-processing

My other task during the project was to write a program for post-processing and visualisation of the results. MITgcm provides some scripts for plotting the results in MATLAB, however most were not compatible with my modified 24-tile-cubed-sphere configuration used concurrently with the NetCDF package, so in the end I wrote almost all plotting and post-processing scripts from scratch. The most important part of the post-processing is projection of the cubed sphere data onto a regular lat-lon grid. This is required for visualisation and also zonal (along longitude) averaging of the data. This enables exploration of phenomenas such as temperature waves, jet streams, easterlies and westerlies winds, cells of vertical atmospheric circulation and others.

Unlike the lat-lon grid, the cubed sphere grid causes some problems with its unevenly distributed grid points and requires scattered data interpolation for further post-processing. This is a computationally challenging task, and even more so when the interpolation must be performed on every single snapshot and every single variable. Fortunately, only 2D interpolation is required as each vertical level can be processed separately – sequentially or in parallel.

As a part of my project I produced a MATLAB application that uses the Parallel Computing Toolbox to postprocess and plot all of the simulation results in parallel. I modified the code to be compatible also with Octave and its Parallel package. The scattered data interpolation is however significantly slower in Octave. I used the MATLAB Compiler to produce a standalone application that can run even on clusters like fionn.

Results

The major driving force of atmospheric circulation is solar heating. On average, this is largest near the equator and smallest at the poles. The atmospheric circulation transports energy polewards, reducing the equator-to-pole temperature gradient.

In our simulations, we used the so-called Held–Suarez configuration. This configuration represents an idealised atmospheric circulation, where land-ocean contrasts have been removed, as have all the mountains and all other zonally-varying boundary conditions and forcing. In other words, the planet is perfectly symmetric. The figure 2 shows the forcing power at different vertical levels as a function of the latitude. The northern and southern hemispheres are identical, meaning that the forcing

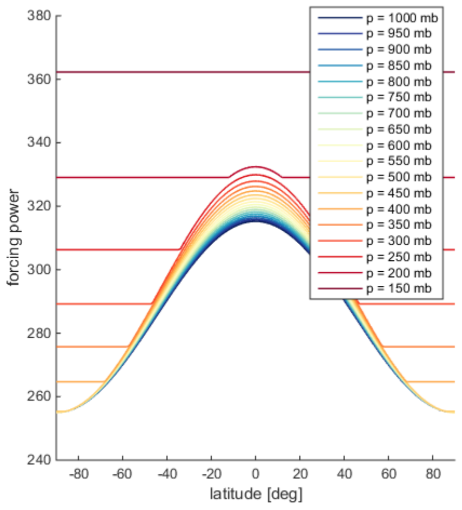


Figure 2: Forcing mimicking sun's radiation: power as a function of latitude. Each line specifies one pressure level ranging from 1000 millibars (ground level) to 150 millibars (topmost level). Generally, the forcing is highest at the equator and lowest at the poles with northern and southern hemisphere being identical. For higher levels, the difference between the equatorial and polar forcing weakens, with the topmost level having a constant forcing.

actually mimics the sun shining at perpetual equinox (e.g., March 22).

The driving force (the sun's radiation) sets up a strong north-south temperature gradient. The atmospheric circulation attempts to reduce this by transporting energy polewards. Initially, the transport is using purely mean-meridional circulations, but as the flow strengthens, eventually it becomes unstable, and breaks down into the turbulent wave-like pattern of cyclones and anti-cyclones that we are used to seeing on weather maps or satellite images.

So while the forcing and boundary conditions are zonally symmetric, the final flow is highly asymmetric, and fully 3-dimensional. Over long time-averages, however, these instabilities become somewhat organised into patterns that maintain "jets", and latitudinal bands of surface westerlies or easterlies. Such organisation is controlled by external parameters such as rotation rate and planetary radius, and our overall goal was to identify how such control operates.

In the following section, several figures showing an example output of the model are shown. As an output of our project, several movies have been generated showing the fields evolving in time. These movies can be accessed on YouTube.

The displayed output variables of the model in the Held–Suarez configuration include the potential temperature and three components of wind velocity. These components are

- *zonal velocity*, with a plus sign eastwards, commonly denoted U,
- *meriodinal velocity*, with a plus sign northwards and commonly denoted V,
- *vertical velocity*, with a plus sign downwards and commonly denoted $\omega = \frac{\mathrm{d}p}{\mathrm{d}t}$.

The variables are visualised in following projections, each revealing different attributes and phenomena.

The spherical projection, used in the movies and also in the title picture, shows a scalar variable field (such as the potential temperature or a single velocity component) at defined level. The variable value is represented by colour. Its advantage is that it depicts the planet "as in real life situation", however the perspective hides some areas from the viewer. Another advantage is, that the the projection shows the cubed sphere discretitsation and therefore does not require any postprocessing, such as interpolations etc.

The latitude-longitude mercator projection, plots also a scalar variable field at a specified vertical level with its value represented by colour. Compared to the spherical plot, it does not hide any data, however it does not preserve scale, with the effective size of the polar regions being larger than the equatorial regions. In this projection, we also plot the zonal and meridional velocity as arrows with the value specified by the arrow size.

The latitude-pressure projection, enables us to see the pressure dependency of the depicted variables. In our plots, the data in this projection are always longitude averaged. This is very convenient together with the Held–Suarez configuration, as its forcing and all boundary conditions are zonally symmetric. The value of the plotted scalar field is again visualised by its colour.

Time averages can be added to all of the mentioned projections. In general, the time-averaging enables us to see more the long-term nature of the climate rather than the short term weather. The key is to select the right averaging time-span. With the zonally symmetric Held–Suarez configuration, that is quite easily achieved by selecting the shortest time, when all the zonally varying features of the field disappear.

The first group of images (composing the figures 6 and 4) shows the model output in standard Earth-like configuration. These figures presents the capabilities of the model and also provide some reference for the next set of figures. The second group of images, composing the figure 5 presents the capabilities of the developed slowevo package as the model was evaluated with a continuous change of selected parameters.

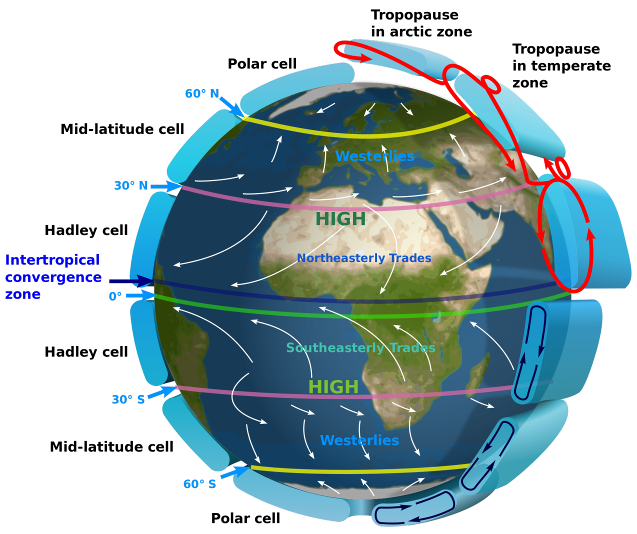
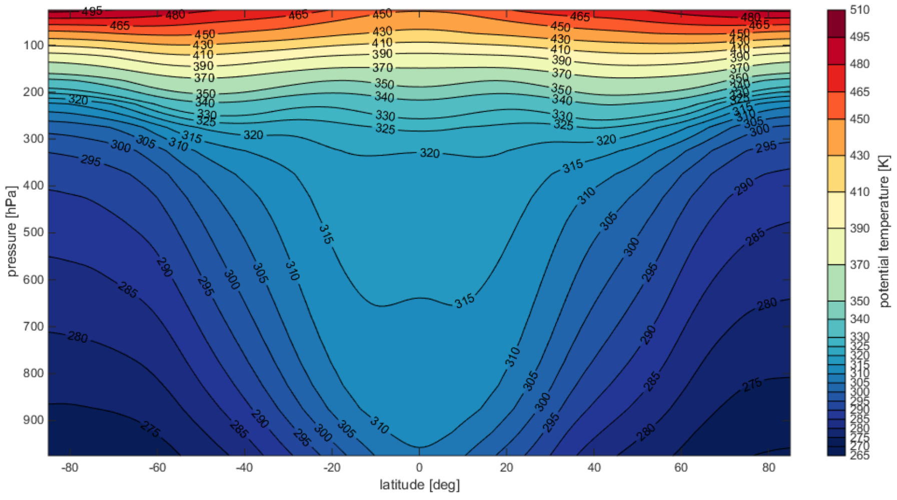
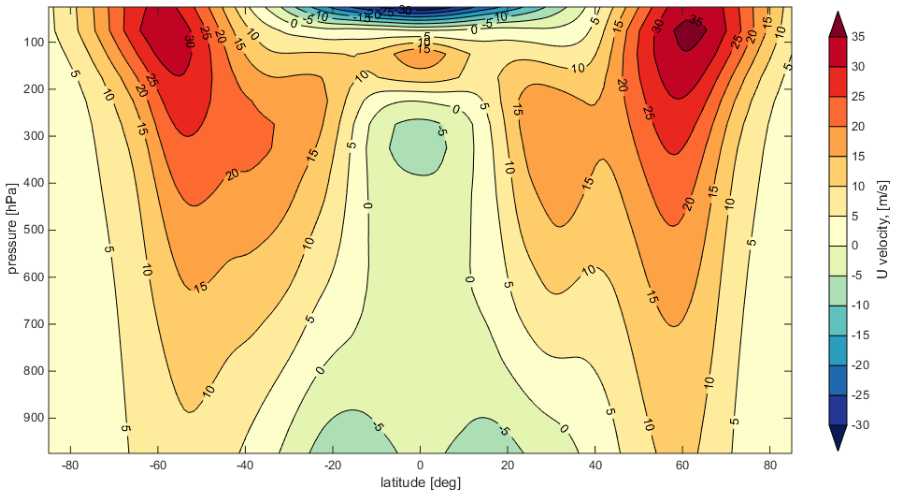
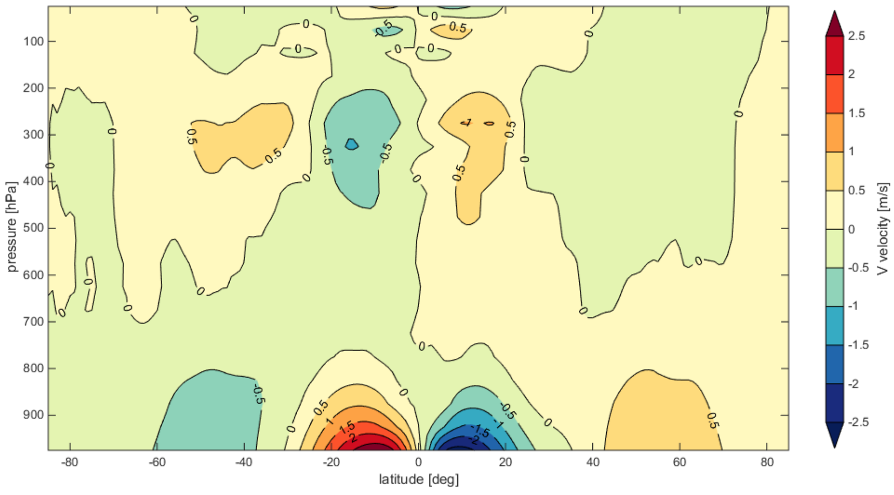


Figure 3: Atmospheric circulation. Highly idealised atmospheric circulation somewhat similar to the Held–Suarez configuration. For reference to the figures of simulation output see the Hadley cell, the mid-latitude (or Ferrel) cell and the polar cell and corresponding trade winds, westerlies and easterlies. The jets, though not depicted on this figure, are located at about 60 degrees (polar jets) and 30 degrees (subtropical jets). Again, their location also corresponds to the location of the cells. Public domain image by NASA.







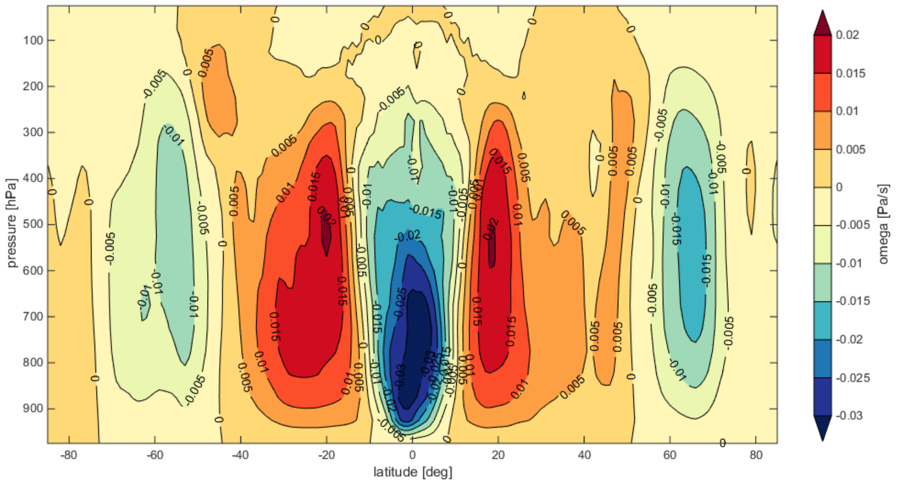


Figure 4: Reference simulation

Various projections, time-averaged over 10 days, after 6 months of simulation. Simulation parameters representing Earth-like planet: rotation period = 86400 sec, radius = 6370 km, gravity = 9.81 m/s^2.

The first figure depicts the zonally averaged **potential temperature** in latitude-pressure projection. The temperature profile is flattening in the upper levels. The highest temperature gradient at lower levels corresponds to the latitude of jets' position – see next figure.

The second figure depicts the zonally averaged **zonal velocity** in latitude-pressure projection. Two strong jets (high altitude, high velocity eastwards wind streams) can be seen at 60 degrees latitude. This with good agreement with prediction, the Earth also has two similarly positioned jets called polar jets. The weaker subtropical jets at about 30 degrees latitude can also be seen in the picture, although not as clearly as the polar jets. A hint of another jet can be observed above the equator, forming so-called equatorial super-rotation. Equatorial super-rotation is a valid climatological regime, which, however, does not form on Earth.

The third figure depicts the zonally averaged **meridional velocity** in latitude-pressure projection. Two strong cells of near-surface level winds can be seen around the equator. These are called *trade winds* and their direction is towards the equator – resulting in opposite sign depicted in the figure – and westwards. The westwards component of their velocity may be seen on the previous figure as two slightly negative cells at the same position. On both sides further from the equator, westerlies winds are represented by cells of winds in pole-direction. Their location corresponds to the location of winds fairly strong east winds, which can be seen just below the jets on the previous figure. Another set of winds, so-called easterlies should be present at polar regions, however, in this picture, these winds can not be seen.

The fourth figure depicts the zonally averaged **vertical velocity** in latitude-pressure projection. The positive sign means velocity towards the surface. We can observe the so-called Hadley cells – upwards wind motion above the equator and downwards motion at about 30 degrees. Another cells, the so-called Ferrel cells can be seen between 30 and 60 degrees. The polar cells, located at the polar regions, can not be seen very well, but at least the sign of the wind at the polar region is as it was predicted – positive.

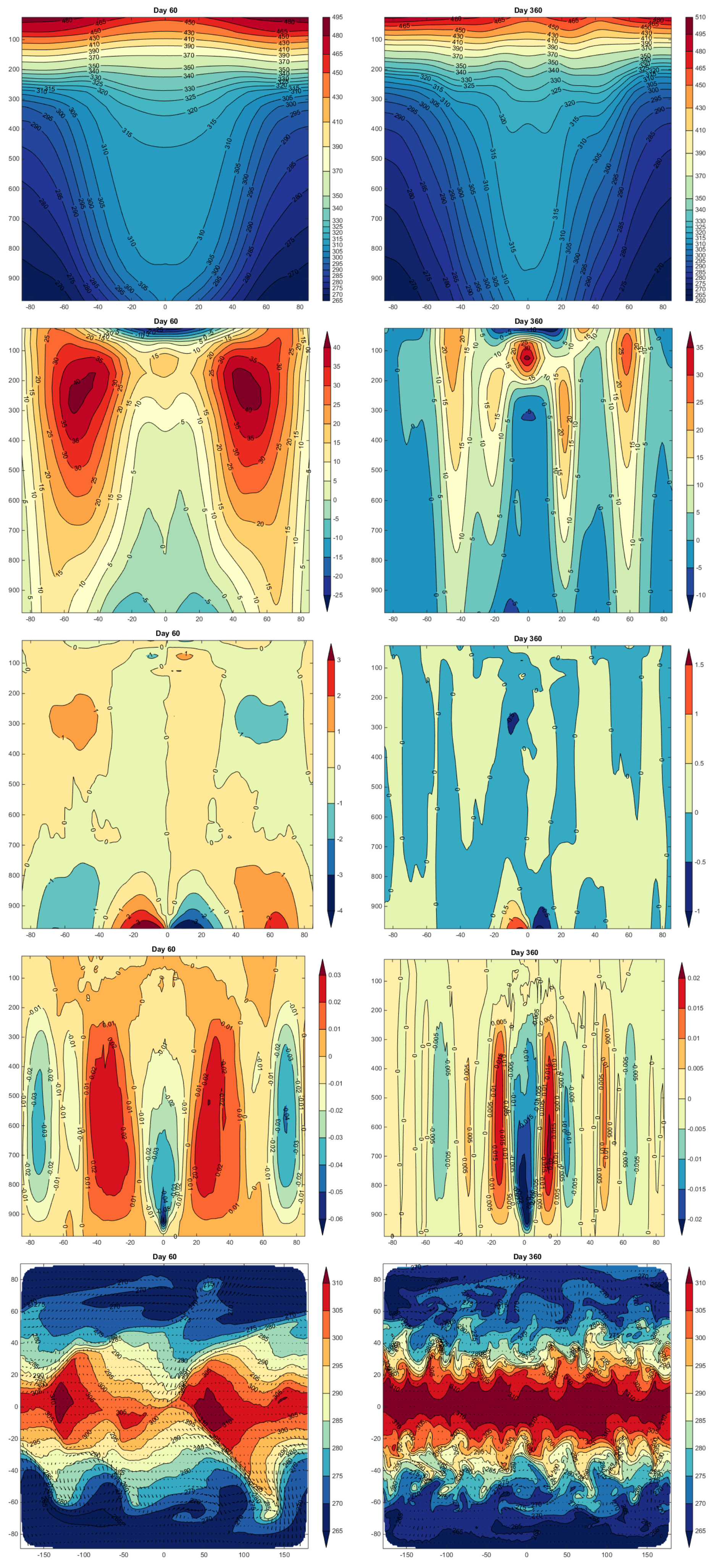


Figure 5: Slow evolution of rotation rate

Various projections, time-averaged over 30 days, after 60 (left-side pictures) and 360 days (right-side pictures) of simulation. Simulation parameters representing a planet with Earth-like radius = 6370 km and gravity = 9.81 m/s^2. The rotation period was continuously decreasing from double the Earth's (172 800 sec) to half the Earth's (43 200 sec) between the day 60 and 300.

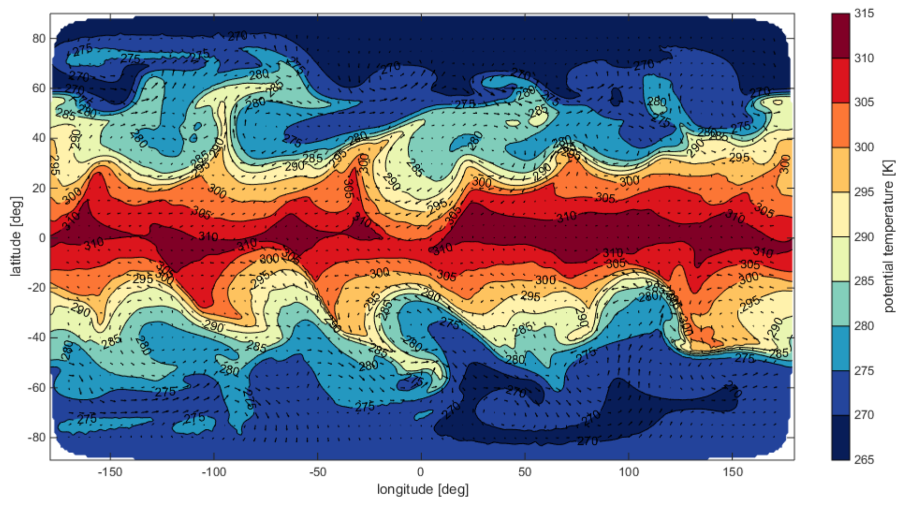
The first row depicts the zonally averaged **potential temperature** in latitude-pressure projection. Two major responses to the increasing rotation rate may be observed. First, the high temperature "valley" becomes more narrow, resulting in colder climate further from the equator. Second, the temperature profile shows a local maxima, i.e. "hills" above the equator. This is connected to the newly formed equatorial jet, the so-called equatorial super rotation (see next figure).

The second row depicts the zonally averaged **zonal velocity** in latitude-pressure projection. Two strong jets can be seen on the left picture (low rotation rate) while multiple jets can be observed on the right picture (high rotation rate). During the simulation, the equatorial jet is becoming stronger, being the fastest of the jets at the simulation end. A tendency for the other jets to become thinner and move closer to the equator can be seen as well.

The third row depicts the zonally averaged **meridional velocity** in latitude-pressure projection. As a response to the increasing rotation rate, the winds present in the case of Earth-like simulation (trade winds, westerlies, easterlies) becomes weaker or almost non-existent. Notice also the lower absolute value.

The fourth row depicts the zonally averaged **vertical velocity** in latitude-pressure projection. The main response to the increasing rotation rate lies within the formation of new cells of air circulation. Consequently, the cells becomes also smaller. Again, the absolute value is lower resulting in weaker circulation.

The last row depicts the latitude-longitude projection of surface temperature. This time, the pictures are snapshots, not time averages. The response to the increasing rotation rate is mainly the number of temperature waves and their pole-wise reach. The slower rotation rate results in fewer waves – just 2 or 3 can be distinguished in the left picture, while the number of temperature waves in the right picture is clearly much higher. The pole-wise reach of the waves gets smaller with increasing rotation rate.



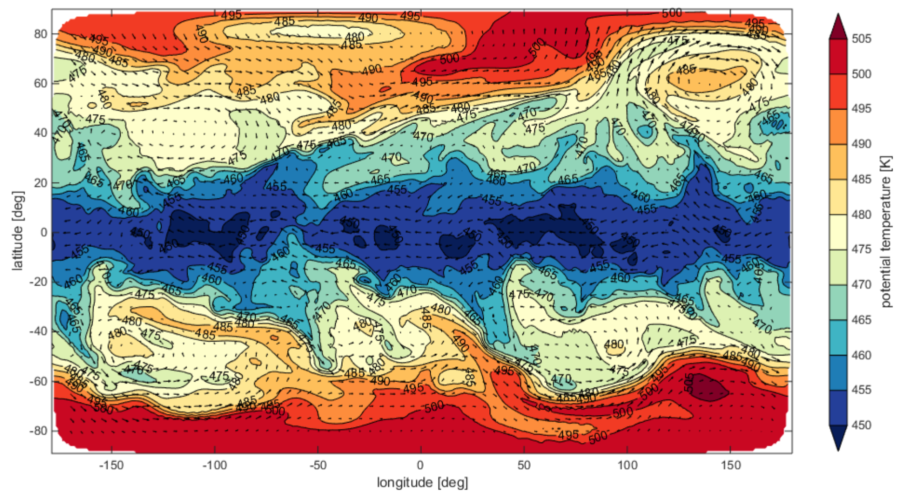


Figure 6: Reference simulation

Potential temperature in latitude-longitude projection, snapshot (not time-averaged) after 6 months of simulation. Simulation parameters representing Earth-like planet: rotation period = 86400 sec, radius = 6370 km, gravity = 9.81 m/s^2.

The first figure shows the temperature at the surface level (p = 1000 hPa). Approximately 6 temperature waves can be seen, which is in good correspondence with prediction.

The second figure shows the temperature at the topmost level (p = 100 hPa). It can be observed, that the temperature is higher at the poles than at equator.

Similar images may be easily produced for every other pressure level, other variables are available for plotting as well.

During our experiments, multiple simulations with different configuration of evolving parameters were produced, such as a simulation of almost stationary planet evolving into very fast rotating planet, a small Moon-sized planet into a huge Jupiter-sized planet and other. This document should, however, just present our work and illustrate the model capabilities. There are wide opportunities for extensive research of parameters dependency with more complicated models.

Conclusion

All three stages of my work, the MITgcm model setup, customised post-processing and creation of the slow evolution package were successfully finished.

The created methods were implemented in a way, that enables usage of these methods with more advanced (and therefore more physically interesting) models.

The methods were tested thoroughly on a simple configuration – the Held–Suarez configuration – and all results were in a good agreement with predictions.

To summarise the physical response of the model to the continual change of parameters, the rotation rate, the radius and the gravity, the greatest effect came from changing of the rotation rate. The low rotation rate resulted in more relaxed atmospheric flow, and made the energy transport from the equator to poles easier, while high rotation rate resulted in more constrained flow. The planet radius affected mainly the number of cells of horizontal circulation. The gravity did not seem to have a great effect with the current configuration. The interval of gravity values, within which study was performed, was however narrow, and more extreme values may result in more dramatic response of the model.

References

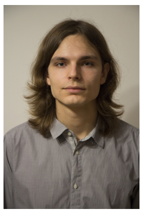
[1] Held, Isaac M. and Suarez, Max J. (1994) A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models *Bull. Amer. Meteor. Soc. 75:1825–1830*

PRACE SoHPCProject Title
Interactive Manipulation of Weather Regimes during Continuous Climate Simulations

PRACE SoHPCSite
ICHEC, Ireland

PRACE SoHPCAuthors
Ondřej Ticháček, [FNSPE CTU Prague,] Czech Republic

PRACE SoHPCMentor
Enda O'Brien, ICHEC, Ireland


Ondřej Ticháček photo

PRACE SoHPCContact
Ondřej, Ticháček, FNSPE CTU
E-mail: ondrejtichacek@gmail.com

PRACE SoHPCSoftware applied
MITgcm, netCDF, ncview, nco, MATLAB, Octave, grads

PRACE SoHPCMore Information
mitgcm.org, ucar.edu/netcdf, mathworks.com, iges.org/grads

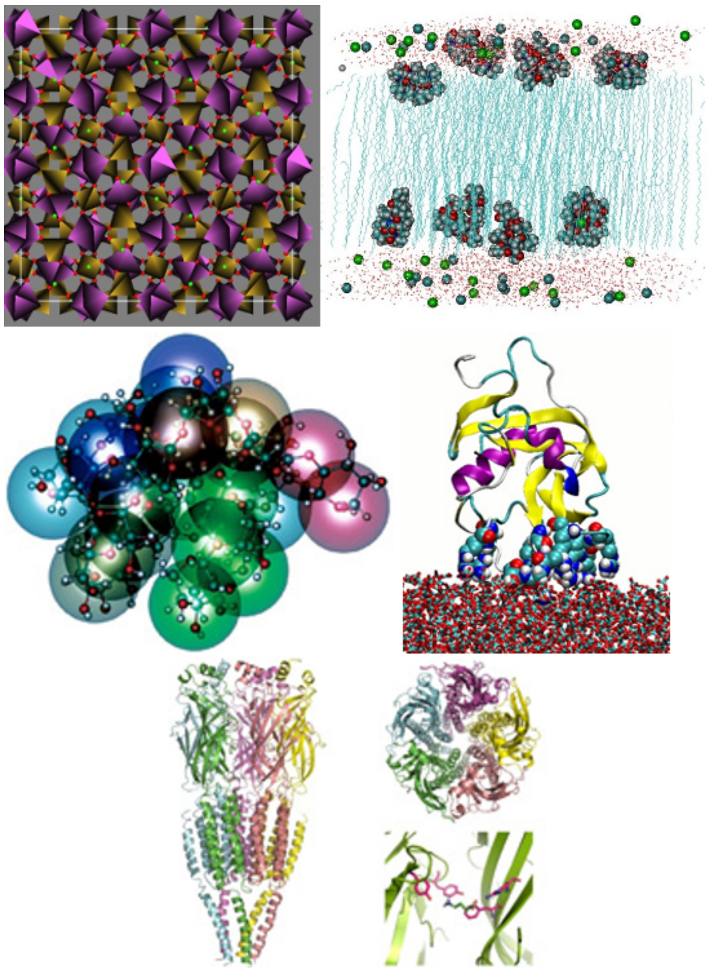
PRACE SoHPCAcknowledgement
I would like to thank my supervisor, Enda O'Brien for all advice and guidance he gave me. I would also like to thank Adam Ralph and Marco Grossi for their kind help whenever I needed it and Simon Wong for coordinating my stay in ICHEC. I would also like to thank Leon Kos and everyone from PRACE for making the Summer of HPC possible.

PRACE SoHPCProject ID
1510

Enabling Real-Time Visualisations of Molecular Dynamics Properties in DL_POLY_4 (Proof of Concept)

Real-Time View in DL_POLY_4

Paolo Grossi



DL_POLY_4 is a general purpose classical molecular dynamics simulation software; this project has the goal of extending package to enable 2D/3D visualisations of molecular simulations in real-time, representing a significant advance.

As part of the simulation of molecular interactions you can take advantage of the incredible power of modern supercomputers, to study better and faster how these interactions take place. Thanks to the availability of ICHEC (Irish Centre for High-End Computing) and the access to the supercomputer Fionn indeed I have the unique opportunity of being able to develop a framework for 2D/3D visualisations of molecular simulations in real-time.

To do that I must first develop a client-side tool for scientific visualisation; for this purpose together with my coordinator I selected ZeroMQ to make the connections between languages and R for software implementation. These software have been selected as multiplatform and easily available since open source, in view of possible future improvements. After all that, I can approach with the source code of DL_POLY_4.

Project Description

ZeroMQ: Code Connected

One of the first tools that I began to study was ZeroMQ which, citing the guide, aims to be a connector between different languages in order to convey information within ad-hoc prepared socket. This is quite convenient as it is open-source, it is widespread among many languages (and is easily extensible also to those for which there are no implementations) and through a series of compromises accurately described it allows to realise many communication patterns that don't overlap code made, but just placed nearby, keeping the reading clear and intuitive.

ZeroMQ can take advantage of a strong network of online enthusiasts and developers created with the passage of time and is now at version 4.1.2, which was released in June. Next to the main there are many branched projects from secondary branches, to try to cover as many languages as possible: to date we can count supports considerable for C, PHP, Python, Lua, C++, C#, CL, Delphi, Erlang, F#, Felix, Haskell, Java, Objective-C, Ruby, Ada, Basic, Clojure, Go, Haxe, Node.js, and many others.

R for Statistical Computing

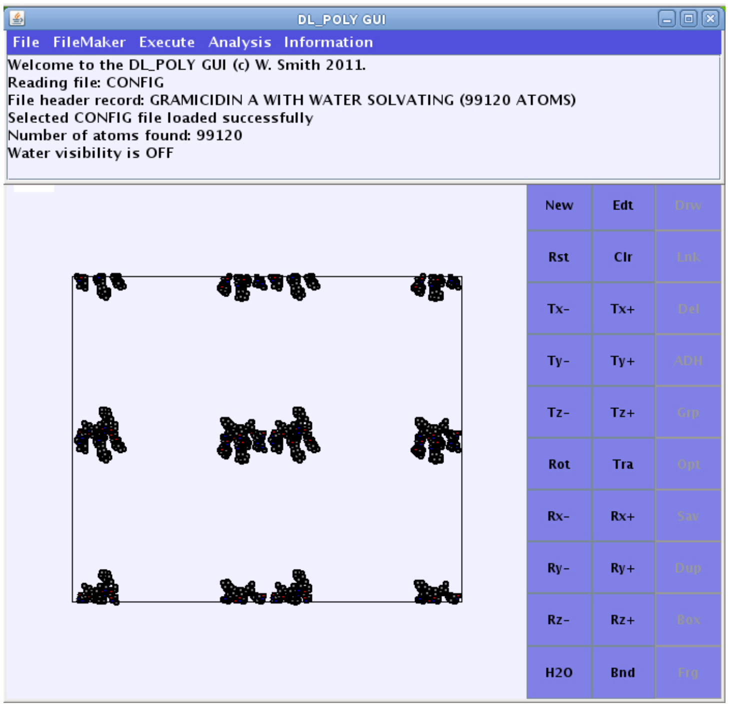
R it's a free software environment for statistical computing and graphics supported by a very energetic and active community; but why is this particular tool the one I picked? Initially my coordinator, Adam and I were oriented toward a different choice; but since R fits well with the project which uses ZeroMQ and since it allows, with some open-source libraries that are well developed, getting the results via the more or less complex graphics statistical analysis easily, I didn't think too much prior to in moving forward with our plan B for my SoHPC.

R package for ZeroMQ

Thanks to the open-source project called rzmq I can in fact use a ready-to-use package in order to join the graphics capabilities of the R programming language with the ease of use of socket-like connections made available by ZeroMQ.

DL_POLY_4

DL_POLY is a general purpose classical molecular dynamics simulation soft-



```
Welcome to the DL_POLY GUI (c) W. Smith 2011.
Reading file: CONFIG
File header record: GRAMICIDIN A WITH WATER SOLVATING (99120 ATOMS)
Selected CONFIG file loaded successfully
Number of atoms found: 99120
Water visibility is OFF
```

Figure 1: Visualisation of Gramicidin A inside DL_POLY

ware developed at Daresbury Laboratory. DL_POLY_4 general design provides scalable performance from a single processor workstation to a high performance parallel computer. It is supplied in source form under licence and can be compiled as a serial application code, using only a Fortran90 compiler.

Initially the installation of DL_POLY on Fionn was completed following the procedures set up for default by software developers but in the last days we have decided to change into a mode called, precisely, Fionn ICHEC to achieve better results. However, this was due to problems that we couldn't resolve within the deadline of the project.

Fortran90 binding for ZeroMQ

Because the source code of DL_POLY is written in Fortran90, I had to look around to find a way to be able to exploit the potential of ZeroMQ within this format: it doesn't exist, neither through the official support nor using open-source projects, a direct connection between the two environments. So I created an official support for ZeroMQ written in Fortran90, based on open-source project found online f77_zmq, which I renamed as f90_zmq_wip during the project time.

Final Product

I realised for the project described: a client written in R for receiving information in real-time while performing the executable of DL_POLY; the server side has been implemented within the source code written in Fortran90 of DL_POLY, particularly within a subroutine that deals with the calculation of statistics and transmitting them to an output file. Alongside of the main project, and for its implementation, it was necessary to also write, as amply described, an extension of the functionality in Fortran90 of ZeroMQ, using what the network has provided but also implementing additional functionality. In particular, few minor changes were required in the management of Publisher-Subscriber pattern used, thanks to the original developer, to fix some bugs.

Discussion & Conclusion

What I produced during the project? It all started trying to analyse the best possible solutions to exploit the project.

The requirements analysis was simple at first glance, but in reality has necessitated some stepwise refinement: it was initially selected VTK (The Visualization Toolkit), a very powerful tool but too complex to be fully understood by a neophyte in so little time. It was decided to tack to a different instrument, such as R, as known from my coordinator and with a scale learning less than its predecessor.

As mentioned earlier, some problems have arisen regarding the visualisation of the final product: in fact, changing some parameters, we are no longer able to see as we could before. Currently we couldn't understand what could be the problem, although it is certainly limited in a particular area of our knowledge, consisting of a few lines of code.

I am satisfied of the work product that I think can be useful in the future for any updates and improvements; definitely solve the problem of the visualisation is more urgent to correct, but I think it is a feasible work in a few days. DL_POLY allows a display mode internally, but it isn't very useful as rather slow and with few options to work with (see **Figure 1**).

Acknowledgements

The opportunity to live and work abroad for more than two months in a cutting-edge facility was unique and amazing: I am eternally grateful to SoHPC for this.

Inside ICHEC I also found some wonderful people as well as valuable colleagues: a big thank you to Simon, Adam and particularly to my brother Marco, who became for a few weeks also a great workmate.

I mentioned just a few names, but everything about the ICHEC to me is worthy of a warm greeting and a genuine thank you: I was welcomed by a family willing to work well, giving great importance to the relationship between people.

Award Statements

Why should I win the HPC Ambassador Award? And what about to win the Best Visualisation Award?

For the constant activity of sharing perpetrated either by social media or by writing blog posts, trying to explain with enthusiasm tone what my SoHPC was all around me: so much passion and so much work, but also a lot of fun!

I encountered many difficulties along the way, but thanks to a collaborative and friendly work environment we were able to solve even the most difficult problems: SoHPC for me it was an opportunity to improve in every way I look, and that's why I think I can aspire these awards.

PRACE SoHPCProject Title
Enabling Real-Time Visualisations of Molecular Dynamics Properties in DL_POLY_4 (Proof of Concept)

PRACE SoHPCSite
Irish Centre for High-End Computing, Dublin, Ireland

PRACE SoHPCAuthors
Paolo Grossi, University of Parma, Computer Science Department, Italy

PRACE SoHPCMentor
Emma Hogan, ICHEC, Ireland

Paolo Grossi

PRACE SoHPCContact
Adam Ralph, ICHEC
Phone: +353 1 5241608 (ext 31)
E-mail: adam.ralph@ichec.ie

PRACE SoHPCSoftware applied
ZeroMQ, R, rzmq, f77_zmq, DL_POLY_4

PRACE SoHPCMore Information
See: zeromq.org, r-project.org, rzmq GitHub, f77_zmq GitHub, scd.stfc.ac.uk

PRACE SoHPCAcknowledgement
I would like to thank PRACE, ICHEC and Summer of HPC for the opportunity to use their precious resources.

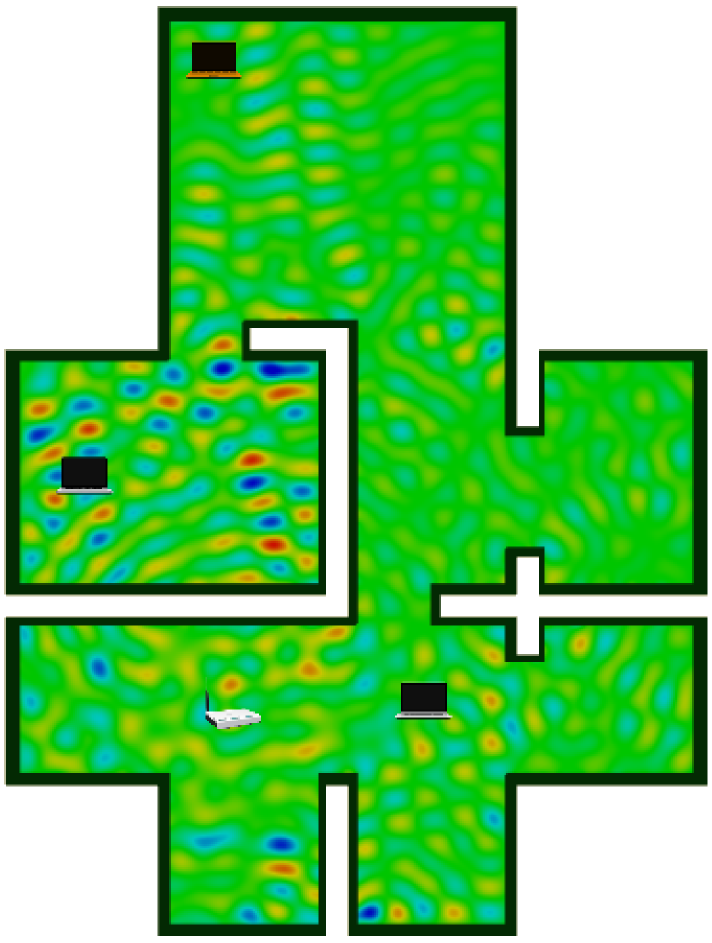
PRACE SoHPCProject ID
1511

Development of a workflow for parallel modelling of the Wi-Fi signal propagation in buildings using the boundary element method

BEM for Wi-Fi

Sergio Afonso

The boundary element method (BEM) is a numerical computational method for solving partial differential equations. The BEM4I software has been used to apply this method on the simulation of the Wi-Fi signal propagation inside buildings. This project will go through all stages of numerical modelling and the development of a workflow for parallel modelling of the Wi-Fi signal propagation.



The use of the boundary element method makes it possible to reduce a problem to just the boundary of the computational domain; in this case the walls of the buildings. This avoids the need to process volume meshes and provides the possibility of working with exterior problems, not only closed spaces. Also, the number of elements to deal with is much smaller when only the boundary is processed. That, the fact that BEM4I is parallelized with OpenMP and MPI, and that its code is explicitly vectorised, make this tool very efficient. The library is extensible and contains other modules for solution of different kind of problems.

This project consists in the creation of tools that complement the BEM4I workflow to make its usage and visualisation of the results easier. One of the tools is responsible for creating the 3D meshes that BEM4I needs as input from a 2D specification of the corners of the building, the height of the walls and the resolution of the mesh. The other tool is a video game in which players have to find the place with the best connectivity in a building, given the position of the router, and find the optimum place to locate the router so that the maximum area is covered. This game will serve as a showcase of BEM4I and as a way to bring HPC to the general public.

Flat Mesher

The first half of this project consists of the *Flat Mesher* utility. This is a tool that turns a set of points on the plane into a three-dimensional triangular mesh of a flat, or building with only one floor. This tool is composed of two parts: A C++ library and a Graphical User Interface (GUI) application. This architecture was selected over a single monolithic application to allow for the creation of many other different tools in the future that rely on the library to make transformations between 2D points and 3D meshes.

Both programs were developed in accordance to the C++11 standard, so they are both multi-platform. The only restriction is that the platform has to be supported by the Qt library in order for the GUI to work.

The library

The C++ library is composed of a set of classes for representing and manipulating basic geometric entities, such as points, lines, triangles and rectangles, blueprints of flats (the set of 2D points) and 3D meshes formed of triangles. It is able to read the input information (2D points, height of the walls and size of each triangle) from text files with a specific format and it is able to output the 3D meshes in both VTU and BEMGEN formats. The design of the library allows the easy addition of new mesh formatter classes, to export meshes into other different formats.

Some parts of the conversion between blueprints and 3D meshes have been implemented in parallel by using OpenMP. OpenMP is an Application Programming Interface (API) for C/C++ and Fortran that allows the programmer to parallelise native code by using high-level directives. This means that the program can calculate multiple different parts of the solution at the same time and most of the code needed to make this possible is provided by the API, so it's much faster and easier to develop.

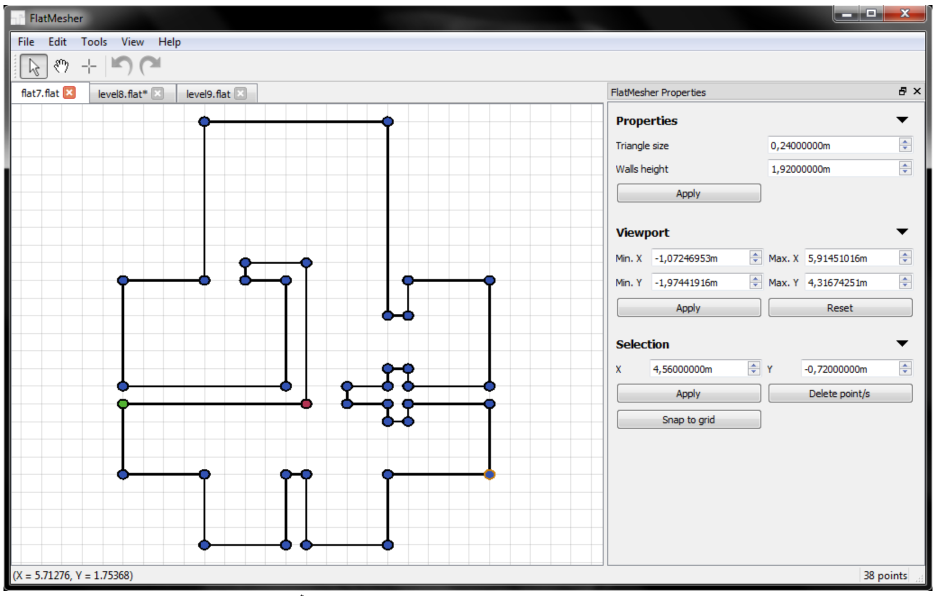
The part of the conversion that is executed in parallel is the creation of each wall and the ceiling, which is also the part that turns out to be the bottleneck of this computation. The creation of the floor sub-mesh is done by copying the ceiling mesh, so it is a fast operation. Each wall corresponds to a segment in the input blueprint.

Once all the walls, ceiling and floor sub-meshes have been created, they are all merged together in a single mesh which is the solution. There is a problem, though. Meshes are represented as a list of 3D points and a list of trios of indexes that refer to positions in the list of points. Each one of these trios is a triangle in the mesh. The problem is that points must not be repeated in the mesh, so the merging has to avoid these repetitions in an efficient way. In this case the solution was to leave out in the creation of each wall the last column's points, which are actually being created in the next wall. When merging, the few triangles connecting each pair of walls are created.

To avoid having to search for the equivalent point index of each point that is on top of a wall when creating the ceiling sub-mesh, and to also avoid the need to finish the creation of the walls before starting the creation of the ceiling, points in the walls were ordered in such a way that their indexes could be determined in advance, hence removing the need for any kind of search altogether.

The GUI

Even though a command-line tool was created to take advantage of the possibilities the library offers, a GUI was thought to be very handy in order to create more easily the flats blueprints than directly working with text files. Since the library to which we had to link was a C++ library, the easiest choice of programming language to develop the GUI was C++, so that's what we chose. We opted for using Qt, a multi-platform application framework to build the GUI.
The application lets the user to load and save *.flat files* (the ones where the blueprints are specified), export them as meshes, inspect them for errors and modify them in different ways. Each loaded flat is displayed and can be interactively edited by dragging, adding or removing points, splitting segments, etc. The viewport can also be changed by using the *hand tool* or by zooming. The appearance of the GUI with one of the levels loaded and a slice of the generated 3D mesh for the same level can be seen in the following figures.

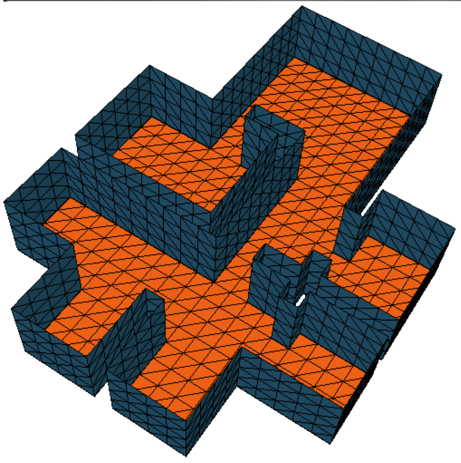


Properties
Triangle size 0,24000000m
Walls height 1,92000000m

Viewport
Min. X -1,07246953m Max. X 5,91451016m
Min. Y -1,97441916m Max. Y 4,31674251m

Selection
X 4,56000000m Y -0,72000000m

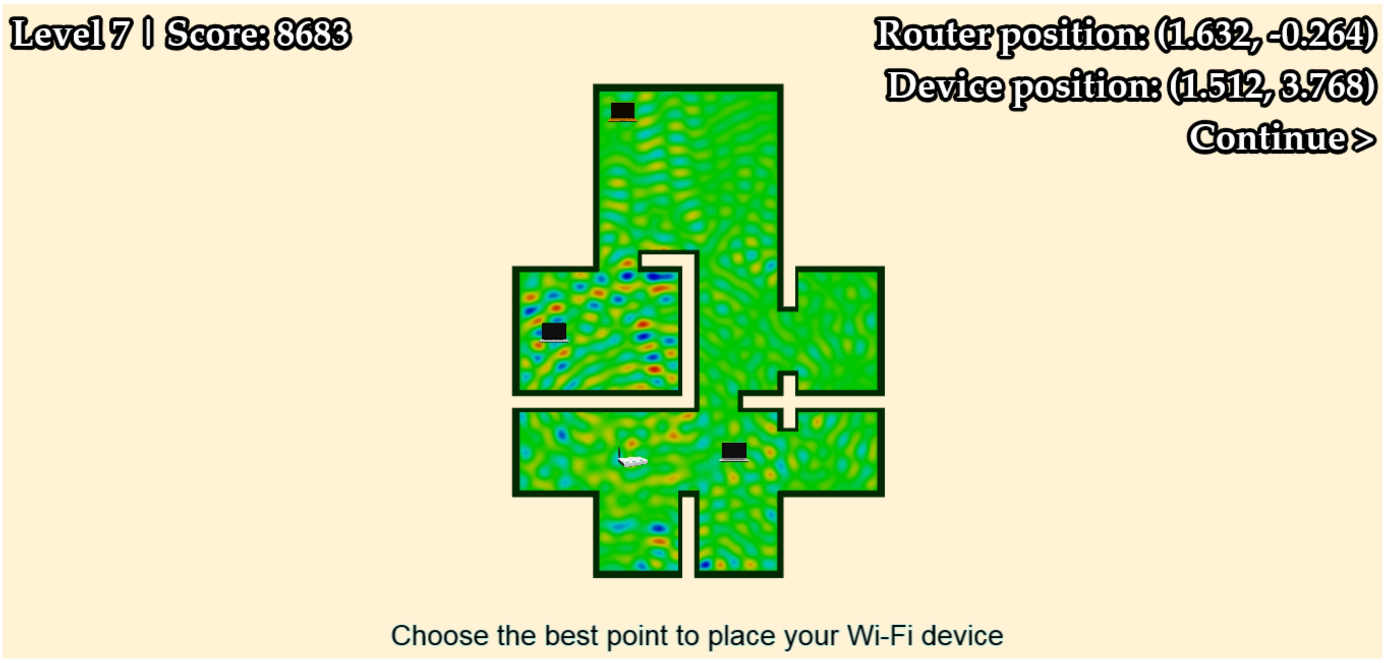
(X = 5.71276, Y = 1.75368)
38 points



The mesh has been rendered with ParaView, a data visualisation software widely used in the High Performance Computing (HPC) field.

Wi-Fi Planner

Wi-Fi Planner represents the second half of this project. In short, it is a web browser video-game designed mainly for desktop or laptop computers, but with support for mobile devices, made of several levels in which you have to first guess, from the available devices and a router location, which is the device that is receiving the strongest signal and then you have to guess, from a set of routers in the same level, which router gives the best overall Wi-Fi coverage inside the flat.
Every time the player selects one of the choices, an animation showing signal propagation through the building is played, so they can have a better understanding of the way Wi-Fi waves behave. Right after that they are presented with a game screen that lets them know how good their decision was compared to the best possible choice, by displaying a rating that ranges between zero and five stars. They can also see the propagation pattern in different slices corresponding to different heights and evaluate the signal strength on each device.
When all nine levels are finished, the player is prompted to enter their name in order to register his or her achieved score in the highest score list, which is saved locally in the client browser. This leads to a more entertaining game.
All the images (slices or animations) are extracted from the results of executing the BEM4I software using the mesh of each level and the router positions as inputs. The computation was carried out


Level 7 | Score: 8683
Router position: (1.632, -0.264)
Device position: (1.512, 3.768)
Continue >
Choose the best point to place your Wi-Fi device

using 48 nodes of the Salomon cluster at IT4Innovations National Supercomputing Center, making a total of 1152 cores and 6144 GB of RAM. ParaView is used to render the resulting meshes and export them as images. After obtaining all the images, they are normalised by cropping the borders and making the background transparent before including them in the video-game.
The scores are calculated in the following way:

- The minimum score min_{score} you can get after each guess is 100, and the maximum max_{score} is 1000. This can be modified.
- The minimum min_{signal} and maximum max_{signal} signal strengths that you can get from the current possible choices are calculated.
- The signal strength corresponding to your choice is also calculated.
- An exponential function going through the points $(min_{signal}, min_{score})$ and $(max_{signal}, max_{score})$ is created.
- The score you get is corresponding to the signal strength associated to the device or router you chose according to the described function.

This application was entirely developed in Javascript and HTML5, using the rendering library Pixi.js v3, which provides good performance by using hardware rendering when possible and then falling back to HTML Canvas when it is not available. This makes it posible to target a wide range of web browsers and get the most performance possible.

Conclusions

I have presented two different applications with very different goals: *FlatMesher*, which integrates into the development workflow of the BEM4I software and greatly reduces the time and effort needed in order to create input meshes for its simulations, and *Wi-Fi Planner*, a video-game that presents the results of the simulations in an entertaining way to the general public and is expected to attract their attention.
FlatMesher delivers every functionality needed and it does it in a user-friendly way. Every use case was considered during the design and implementation of this application, so that it lets you not only interactively create, save, load and modify your flats in many ways, but it also lets you analyse many properties of the created flats in search for errors, it eases common tasks like selecting all points, inverting the points order or changing the viewport to fit the screen. It also has a command-line interface to make the conversion between flat files and meshes non-interactively. Because of its extensibility in terms of output formats and possible different applications, this tool could also be used outside this specific project, in any other application that needs to create meshes from 2D points or as a fast way of creating meshes.
Wi-Fi Planner is a great way to showcase HPC to the general public, since they can see the possibilities of supercomputing applied to their every day life in a way that everyone can understand. Wi-Fi signal strength is a problem almost everyone can refer to. The scoring system is also designed in a way that competing for the best score is really hard, which will hopefully make people want to try it if they see someone else play. It is also sufficiently configurable to change the set of levels, the colour scheme or the fonts to improve its look.

PRACE SoHPCProject Title
Modelling of Wi-Fi signal propagation using the boundary element method

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

PRACE SoHPCAuthors
Sergio Afonso, University of La Laguna, Spain

PRACE SoHPCMentor
Michal Merta, IT4Innovations / Dept. of Applied Mathematics VŠB-Technical University of Ostrava, Czech Republic

Sergio Afonso

PRACE SoHPCContact
Michal Merta, IT4Innovations National Supercomputing Center /
Dept. of Applied Mathematics
VŠB-Technical University of Ostrava
Phone: +420 596 999 613
E-mail: michal.merta@vsb.cz

PRACE SoHPCSoftware applied
BEM4I, Qt, Pixi.js, ParaView

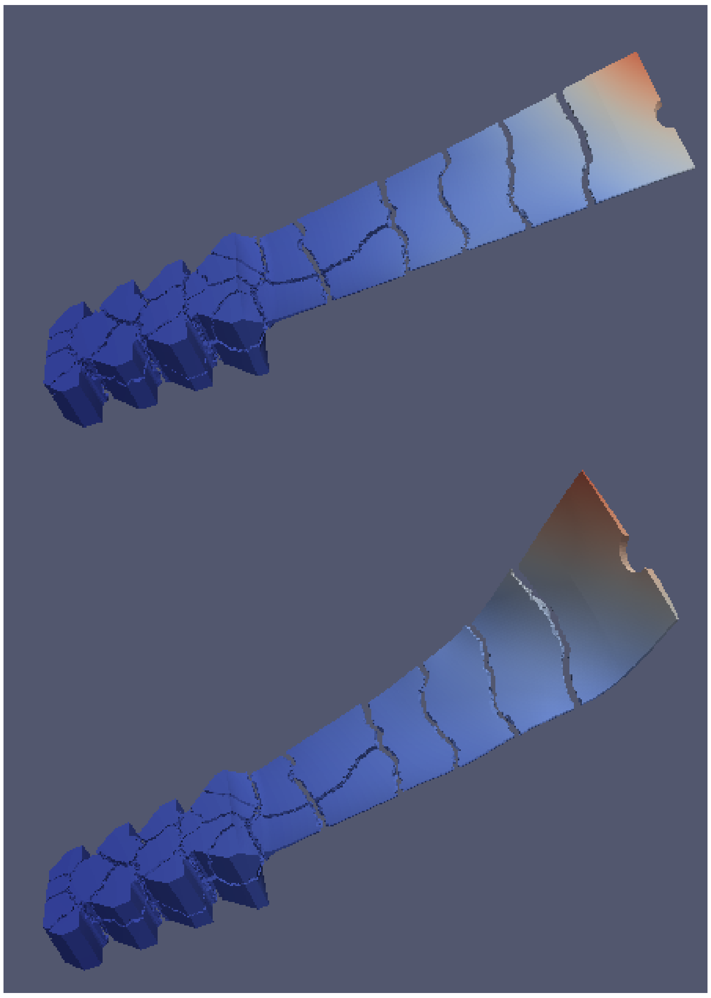
PRACE SoHPCMore Information
http://industry.it4i.cz/en/products/bem4i/

PRACE SoHPCAcknowledgement
I would like to thank Michal Merta and Jan Zapletal for their help and constructive criticism throughout the execution of this project and all the IT4Innovations staff for their kindness.
BEM4I was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

PRACE SoHPCProject ID
1512

ESPRESO Solver for SoHPC

Anthony Bourached



We adapt the code of the **E**xa**S**cale **P**a**R**allel F**E**TI **SO**lver (**ESPRESO**) in such a way that it is linked with Paraview Catalyst during simulation run-time to produce real time, *In Situ*, analysis and visualisation of the simulations. Furthermore, we use IT4Innovations supercomputer, Salomon, to produce visualisations of a fan blade in real time.

This project is based on the visualisation of the results from **E**xa**S**cale **P**a**R**allel F**E**TI **SO**lver (**ESPRESO**). The Finite Element Tearing and Interconnecting (FETI) method is a practical and efficient domain decomposition (DD) algorithm for the solution of numerical partial differential equations. Domain decomposition methods solve boundary-value-problems[1] by splitting them into smaller boundary value problems on sub-domains and iterating to coordinate the solution between adjacent sub-domains. The problems on each sub-domain are independent, this means that we can effectively solve them at the same time. This makes domain decomposition perfect for parallel computing, for a comprehensive explanation of why this is we must discuss the theory of high performance computing.

High Performance Computing

High Performance Computing (HPC) generally refers to the practise of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer. A powerful modern desktop computer typically has eight cores. The purpose of multiple cores is to maximise efficiency of the computer's compute power. It allows the computer to more effectively perform different tasks at the same time.

Maximum program efficiency would be achieved if the program's tasks were divided in such a way that they can be executed by independent cores with, preferably, minimum communication or co-dependency. This practise is referred to as parallel computing. One of the massive aspects of HPC is the use of Supercomputers: massive, highly maintained clusters of computing nodes. Simulations in this project have been produced on IT4Innovations supercomputer, named Salomon.

Parallelism in ESPRESO

From figure 1 we can see how the use of Parallel programming is used to most effectively solve problems with ESPRESO. This is a simulation of a steal block, we shall discuss what ESPRESO is doing with it but first lets discuss its appearance. The gaps that divide the block into many pieces do not physi-

[1]A differential equation with a set of constraints called the boundary values. A solution to a boundary value problem is a solution to the differential equation that also satisfies the boundary.

cally exist in the structure but show us the domain decomposition of the problem. Each sub-domain may be solved in parallel.

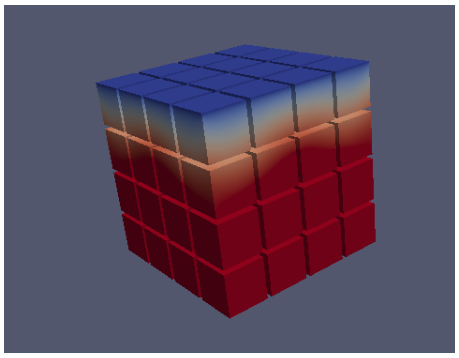


Figure 1: Steal block. Gaps represent the boundary of different subdomains which can be solved independently.

The main object attribute visualised in this project is displacement. We may consider displacement in this context as the distance of each given point of the structure from its original position. Even a rigid structure such as a steal block deforms under force, such as gravity, though this is often far too slight to be detected by the naked eye. However, using Paraview we may warp vectors. This means that we can emphasise the magnitude of that attribute without changing anything else about the structure or appearance of the object. For example, warping displacement or velocity by a factor of two will make the displacement appear double of what it actually is or make velocity twice as fast as it actually is. For our simulation of this steal, as in figure 1 and 2, cube we warped displacement by 4^{12} (4,000,000,000,000).

In Situ Visualisation

The goal of visualisation is often to find desired/important details within a large body of information. As well as getting a general understanding of the structure and attributes of an object on data sets that are far too large and complex for the human brain to comprehend without such tools. Paraview is often used for such visualisations and is used in this project.

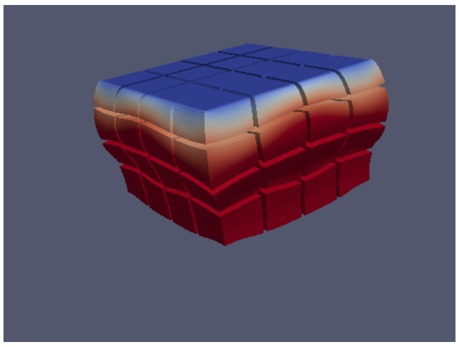


Figure 2: Steal Block. This is the exact same image as in figure 1 (note the colours) except that displacement has been warped by a factor of 4^{12}.

We can see the steal cube squashed under gravity.

We use Catalyst[2] and Paraview to perform *In situ*[3] analysis. It is common to require that our simulations discard most of the data created in order to maximise efficiency. Since, in this case, it is not possible to store data for many simulations, data analysis and visualisation must be performed *in situ* with the simulation to ensure that it is running smoothly and to fully understand the results that the simulation produces. This is opposed to the traditional workflow where simulation would be ran and data output to a disk and then finally analysed and visualised (using, say, paraview).

There are many advantages to *in situ* analysis. First, we can begin the analysis and visualisation process without needing to do any input/output of simulation results- this is an especially important point as simulation power is advancing at approximately ten times the rate of input/output efficiency. Second is that rather than performing our visualisation/analysis on another machine we have the full computational power of the supercomputer available to do this processing. Third, we expect that the the data produced by the analysis/visualisation will be much smaller than that produced by the simulation itself.

For these reasons I believe that co-processing will have a massive role in the future of HPC and thus was great motivation for this project.

Method

My task was create the API (Application Programming Interface) for the ESPRESO that enables it to run with Catalyst and hence visualise the results in real time.

Before I started my project, ESPRESO only enabled post-processing; It output results to a vtk file. **V**isual **T**ool **K**it (vtk) is a file type which is readable by Paraview. Two important data types give the structure of the object: 1) Points: we must have an array of points which outline the shape at, for simple cases, uniform intervals. Each point has three associated values: its x, y and z coordinates. 2). Cells: these will be the most fundamental element of the image.

So far this gives the structure of the object. Furthermore, ESPRESO records attributes, such as displacement which is what we have focused on in our visualisations.

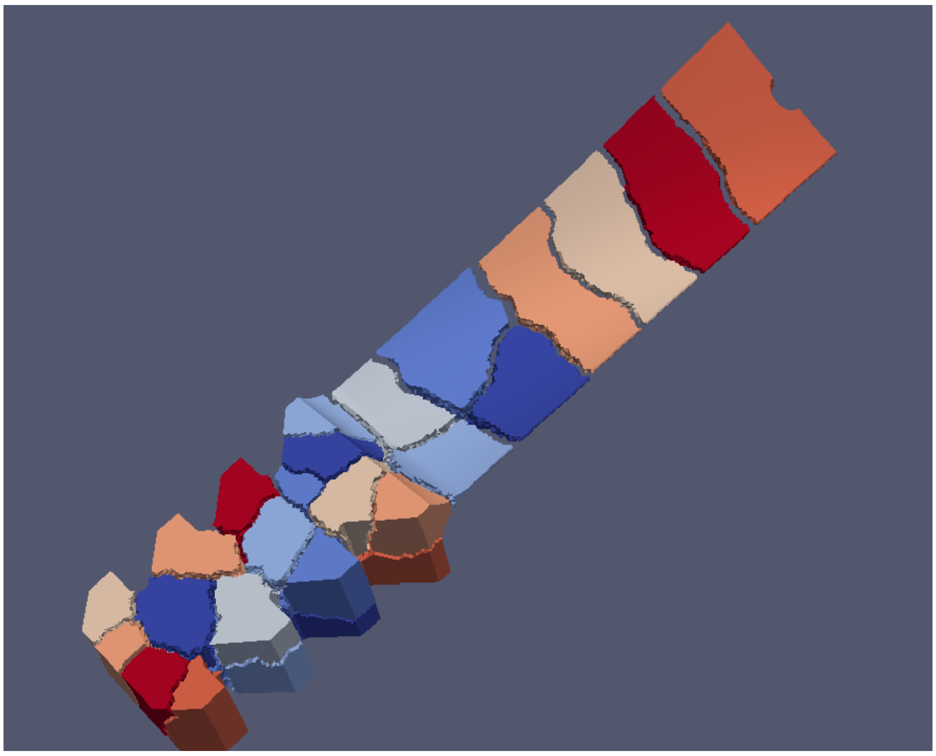
To perform our real time visualisation all these values had to be passed from ESPRESO to Paraview Catalyst by an Adaptor code which constituted the most significant part of the API.

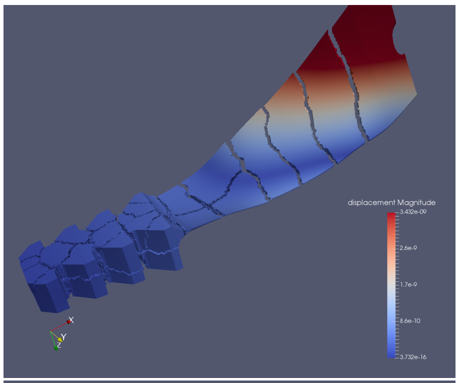
ESPRESO is written in c++ so we also used this language to write the Adaptor code. The structure of the object is unchanging so the Adaptor code could be configured such that we only need to build the grid points and cells on the first iteration, hence only requiring for the values of displacement to be updated at each timestep. This ensured that communication between ESPRESO and Catalyst was as fast as it could be. Already this means that we save hugely on the amount of data communication required each timestep as Catalyst keeps the object structure in its memory.

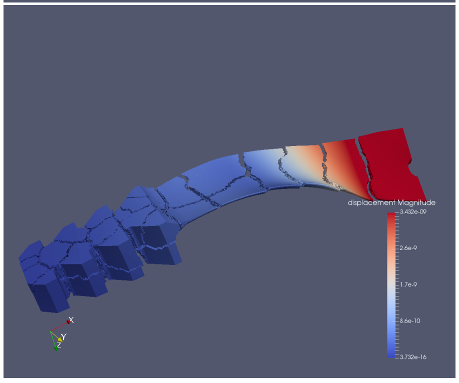
When images are rendered in Paraview from a vtk file one usually uses Paraview to show the exact aspect of the data that is required. For example, one might like to slice the object to show a transverse image of its interior. These tools are infinitely useful and use of them is one of the main motivations for visualisation. A python script was written/exported by Paraview Catalyst which enables us to render a real time image with all our specified Paraview

[2]Catalyst is a 'light-weight' version of the ParaView server library that is designed to be directly embedded into parallel simulation codes to perform in situ analysis at run time. Essentially it enables us to use the same analysis and visualisation tools that are already available for post-processing using Paraview.

[3]*In situ* is Latin for 'on site' or 'in position'. In this context *In situ* analysis refers visualisation and analysis as the simulation is running. This form of analysis can also be referred to as co-processing or co-visualisation.







Similar to the cube, the gaps that look like cracks in the fan blade do not physically exist but show the domain decomposition. This decomposition is emphasised by colour coding in the large image. Displacement is warped in the other two images by ten billion (10,000,000,000). This makes it look like a flag blowing in the wind. These are timesteps 282 and 421 respectively. The first timestep also illustrates the what the fan looks like in real life (with no warped attributes).

configurations. One of the specification of this script was to save a PNG of each timestep which we later used to make AVI videos.

Results and Discussions

As the main focus of the project was to create the API for ESPRESO and Catalyst; results are visualisations produced by the ESPRESO. The images shown in this report were rendered in real time-while the simulation code was running. From these visualisations we saved images for each timestep and later compiled them together to make AVI (Audio Video Interleave) videos.

Our main visualisation was a fan blade as can be seen above. The large image shows the colour coded decomposition of the structure.

The title image shows two identical similar images. The one on top is a fan blade with a colour scheme that illustrates each points displacement from the ideological shape of the object. This displacement is far to slight to see so the only evidence of it in this image is the colour. The image immediately below, however, is the exact same image but with displacement warped by 10^{10} (10,000,000,000). This second image therefore enables us to see what the colour in the first image is telling us about the data.

For simulations that produce large amounts of data, Co-processing is becoming a necessity. It is quite likely that in the next 10-20 years all of the largest simulations will require in Situ analysis in order to not drastically lose efficiency.

When we post-process using Paraview we may replay simulations very easily. This gives us the luxury of being able to take our time specifying what part of the data we want to emphasise. The 'down side' to the massive amount of work we avoid by not recording data with co-processing means that we are not able to view the simulation again (at least not with all the details that were in to original version.) Therefore, we need a way to specify the exact render conditions we want textitbefore we start the simulation. A python script was used in this project to specify the render conditions of the co-processor (Paraview Catalyst). This made it possible to perform co-visualisation without any loss of functionality of Paraview. This means that we can 'have our cake and eat it too' in terms of benefiting from the speed and convenience of co-visualisation/co-analysis while still using all the features of Paraview.

References

[1] Andrew C. Bauer, Berk Geveci, Will Schroeder Kitware Inc. February 2015. The Catalyst Users Guide v2.0 ParaView 4.3.1

[1] Andrew C. Bauer, Berk Geveci, Will Schroeder ...et al The Paraview Guide, A Parallel Visualisation Application *Kitware Inc.*

PRACE SoHPCESPRESO Solver
ESPRESO API for ParaView Catalyst to perform In situ Analysis.

PRACE SoHPCIT4Inovations
IT4Innovations,

PRACE SoHPCAuthors
Anthony Bourached, [association,] Czech Republic

PRACE SoHPCMentor
Riha Lubomir
lubomir.riha@vsb.cz

Anthony Bourached

PRACE SoHPCContact
Leon, Kos, Univerza v Ljubljani
Phone: +12 324 4445 5556
E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied
MPI - MPICH 3.1, ESPRESO, Paraview Catalyst

PRACE SoHPCMore Information
www.paraview.org/in-situ/

PRACE SoHPCAcknowledgement
I would like to extend my gratitude to my mentor Riha Lubomir for many useful meetings and guidelines throughout the project. I was also like to thank Alexandros Markopoulos for helping to explain the structure of ESPRESO.

PRACE SoHPCProject ID
1513

3D Visualisation from CT

Laszlo Kovacs

In this study we demonstrate how to segment the human liver using the HPC environment from the CT images and create a photo-realistic visualisation to support medical treatment being a more precise, faster and with lower risk. Our new real 3D viewer supports the hand-gestures control by the Leap sensor and Virtual Reality by the Oculus glasses. The results are planned to be used in the real medical treatment focused on the lethal liver cancers.

TThe High Performance Computing (HPC) systems play more and more important role in our society thanks to the increasing size of processed data, and the higher computational demands. These aspects could be identified in many areas of the daily life such as medicine, weather report, traffic monitoring or industrial environment. High computational power together with high efficiency of HPC systems is effectively solved by the usage of coprocessors.

The Computed Tomography (CT) is one of the fields which generates big amount of data in medicine. The CT scanning or also so-called X-ray computed tomography (X-ray CT) could be considered as a non-invasive procedure since it allows us to see inside the human body without clinical surgery. The only limitation of the technology is the x-ray radiation dose during the process and its effects on the human body. The risk that comes from the radiation is much lower nowadays thanks to the new types of CT machines. Moreover, a punctual and more easily understandable visualisation of the data can prevent from repeating the scanning intervention.

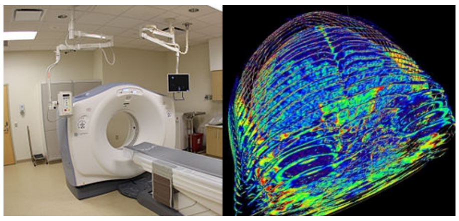


Figure 1: CT machine and the results of the scanning

During the scanning process the CT machine recreates the inner image of the body in sequential axial slices (Figure 1). The CT images are stored in a special Digital Imaging and Communications in Medicine (DICOM) format. This is a well-known and widely adopted standard by hospitals and medical device industries for handling, storing, and transmitting all the necessary information in medical imaging. In the viewpoint of the medical treatments the precise and realistic image processing and visualisation are important since they can obviate the repeated scanning and directly help doctors with the evaluation and decision making process. A typical application of it is the liver carcinoma, which is a very common a malignant epithelial tumour. Based on World Health Organisation (WHO) and other research studies the human cancer is a major health problem worldwide. Over a million deaths per year, which is about 10% of all deaths among the adults, can be contributed to hepatocellular carcinoma all over the world. On the picture below you can see how looks like the hepatocellular carcinoma 2. Regarding this cancer one of the most important information coming from the segmentation is the exact evaluation of the organ volume. Based on this information exact liver resection is made. Without the information the surgery can be lethal. For the punctual liver segmentation and visualisation, we exploit the HPC technology extended with coprocessors to provide the solution. This process consists of four main steps: pre-processing, segmentation, post-processing, and 3D visualisation.,[12]

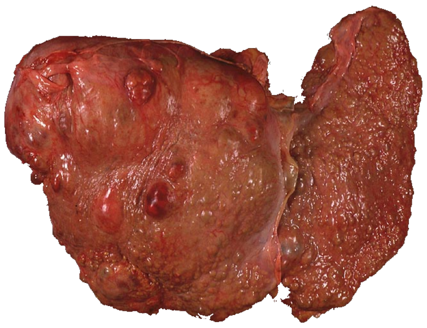


Figure 2: Liver Carcioma

Pre-processing

In the pre-processing step, the first task is to retrieve the data from DICOM standardised medical image data format. In our research we try to use detailed CT images with mutual axial slices of 0.6 mm to support punctual 3D reconstruction. The next important sub-step is the noise reduction by applying the de-noising filters in all the acquired images. During the scanning process, the random noise appears in the images. The noise is tightly connected with the physical principle of the scanning. We have to mention that this is just a half true, because during scanning process many different kind of noise could distort the final output. Nonetheless based on the state of the art studies, the application of our model can suppress the majority of the noise very effectively (Figure 3). The images with suppressed level of noise support the segmentation algorithms to achieve better results. In the current stage of our research we focus on the state of the art methods of Gauss smoothing and BM3D.

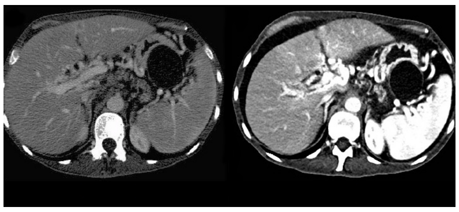


Figure 3: Results of the denoising filter

Before the segmentation step there is a mapping, which is in fact a pixel intensity transformation from CT to Hounsfiled Units (HU). During the calculation we apply the following linear equation, in which the inputs come from only the DICOM format. This mapping is useful, since the different organs (in human body) have different HU values . For instance the bones has typically +700 HU or greater, the liver has +40 to +60 HU, and the air has -1000 HU. As the last sub step we applied image data conversion to store all the image slices in one image vector.

Segmentation

In the case of the segmentation process, we focus on the state of the art k-means method. The generalised algorithm can partition n pixels into k clusters, where k is an integer value and k<n. In our case the k-means algorithm classifies the pixels regarding the similarity features. The pictures below show the segmentation results in the case of one image slice regarding the non pre-filtered and pre-filtered case (Figure 4).

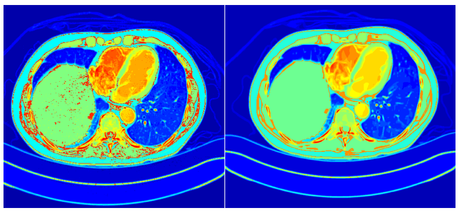


Figure 4: Results of the segmentation without/with denoising

Post-processing

As a post-processing step we proceed to the reconstruction of the surface based on the gained individual segments. These boundary segments are calculated with the flood algorithm. As for the 3D reconstruction algorithm we used the Marching Cubes method. The following picture show of the result of the reconstruction(Figure 5).

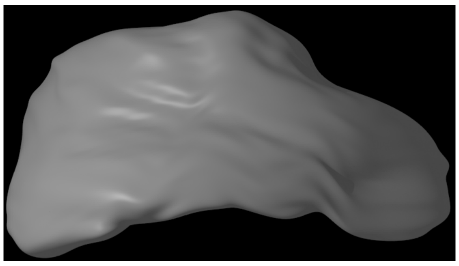


Figure 5: Results of the 3D reconstruction

3D Visualisation

The final step is the creation of the 3D object and its visualisation in a suitable environment based on the segmentation results. With this results we can provide the expected measurement such as volume measurement. During the common work we concentrate on one part of the human body which is the liver. It simplifies the final solution but on the other hand it remains the possibility to easily extend the solution with new organs and features in the future. Considering the high performance and portability demands, we choose the OpenSceneGraph (OSG) as a visualisation environment. The OSG is an open source LGPL licensed high performance 3D visualisation toolkit. It is widely used as scientific visualisation, virtual reality or modelling environment. OSG is written in C++ and supports the wide range of operating systems like Windows, OSX, Linux. The cross platform environment is convenient, since in this way it is possible to show the segmentation results

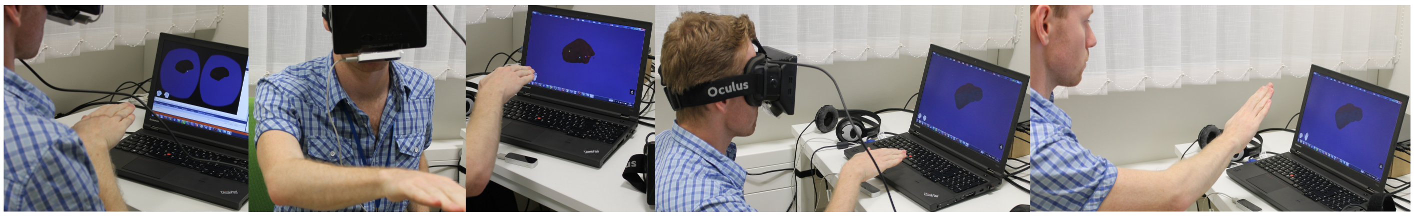


Figure 6: The pictures show how works the 3D visualisation program.

not only on a computer but also on a smart-phone or tablet. This can be useful not only for medical treatments but also during the education of medical students.

With regards to the visualisation part we focus also on the newest real 3D visualisation technology and unconventional LEAP Motion controller for interactive control of the visualised scene. The Leap Motion Controller tracks your hands at up to 200 frames per seconds using two infrared cameras. The field of view of the sensor is 150 degree. We integrate the leap sensor into our viewer to replace the usage of the regular mouse. We add the following features into our program: object grabbing with one hand; object rotating with one and two hands; zooming with two hands; pointer function with one and two hands. While the project focus on real 3D visualisation, we also work with OCULUS Virtual Reality Glasses. This glasses are a virtual reality head-mounted display developed by Oculus VR. The glasses track your head movement and put the viewer in the middle of the scene, while the head mounted display renders the picture to view everything in real 3D from viewpoint of the person. We successfully integrate the Oculus Rift into our viewer. The program can recognise the connected devices and renders the scene on the glasses monitor in a right form. With the help of the glasses you can move your head to follow the object movement more easily. We pay a great attention to create the feeling of the virtual reality as real as possible. We successfully integrate the LEAP sensor and the OCULUS Rift to work together. It means you can use both tool at the same time.

While the project focuses also on the photo realistic post-processing of the acquired data, we chose the Blender project as an open source software to create adequate materials, textures and scenes. It is a modelling and rendering program that can create photo realistic pictures and animations. The 3D model of the liver is imported into the Blender and by help of our own texture materials and by using both Blender's renderers a real photo-realistic liver model is created(Figure 7).

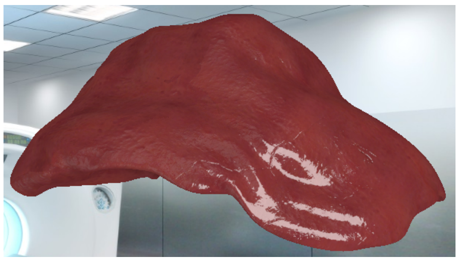


Figure 7: Results of final 3D visualization

Post-processing

This model is exported and used in our own visualiser. Moreover we ready also the the 3D model and the 2D texture for visualisation of the human kidney.

Due to the enormous data processing we used the HPC environment. It was essential in the case of image segmentation and photo realistic visualisation. During our work we used the HPC environment involving the coprocessors NVIDIA Tesla and Intel Xeon Phi.

Results

Our final solution can use the automatic image segmentation of medical images to visualise the liver in a precise and realistic 3D way. The application uses the latest 3D visualisation technology involving the photo realistic post-processing of the acquired data and it is enabled for the users to control the visualised scene by an unconventional LEAP controller interactively even in the virtual reality environment(Figure 6). To achieve the results, we exploit the latest HPC environment involving GPU and CPU coprocessors.

In the future we plan to extend our solution to visualise the full human body based on CT image segmentation. Following this way we have been ready already with the 2D texturising of the human kidney(Figure 8).

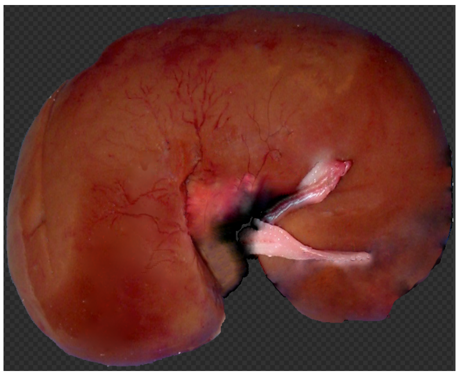


Figure 8: Texture on the 3D kidney

References

[1] Block-matching and 3D filtering (BM3D) algorithm and its extensions. *http://www.cs.tut.fi/~foi/GCF-BM3D/, 2015.*

[2] P. Strakos, M. Jaros, T. Karasek, L. Riha, M. Jarosova, T. Kozubek, P. Vavra and T. Jonszta Parallelization of the Image Segmentation Algorithm for Intel Xeon Phi with Application in Medical Imaging. *Proceedings Of The Fourth International Conference On Parallel, Distributed, Grid And Cloud Computing For Engineering, doi:10.4203/ccp.107.7, 2015.*

PRACE SoHPCProject Title
Medical image segmentation and visualisation

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

PRACE SoHPCAuthors
Laszlo Kovacs, [University of Debrecen,Faculty of Informatics] Hungary

Laszlo Kovacs

PRACE SoHPCMentor
Strakos Petr, VSB-Technical University of Ostrava, IT4Innovations, Csech
Lubomir Riha, VSB-Technical University of Ostrava, IT4Innovations, Csech

PRACE SoHPCContact
Lubomir, Riha, VSB âĂŞ Technical University of Ostrava, IT4Innovations
Phone: +420 59 732 9651 E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied
Matlab, Blender, OpenSceneGraph, Leap Motion, Oculus Rift

PRACE SoHPCMore Information
www.virtouso.org

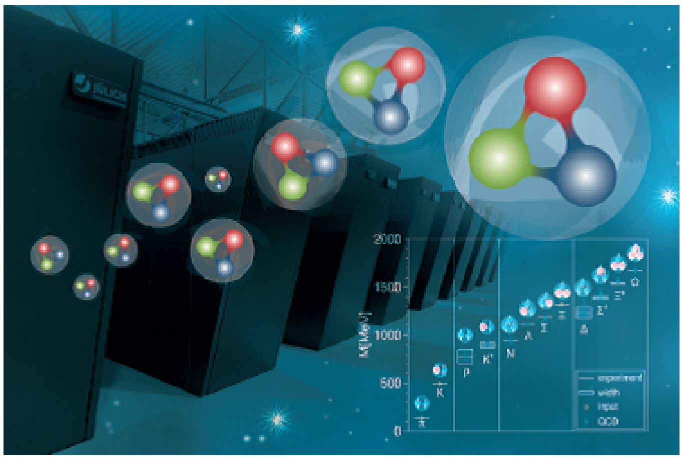
PRACE SoHPCAcknowledgement
I would like to say thank you my mentors about tutoring me and the other colleagues of IT4I at Ostrava about supporting me during two months.

PRACE SoHPCProject ID
1514

Making Quarks Phli Further

Sarah Jenkins

Quantum Chromodynamics is the theory of the strong nuclear force, a fundamental force describing the interaction between quarks and gluons. Simulations of these interactions require the computation of large, extremely sparse linear systems which need a considerable amount of computational power. Therefore, optimisation of current software packages is an important step in improving the accuracy of these simulations by improving the statistical precision at fixed costs.



Quantum chromodynamics (QCD) is the theory of the strong interaction, one of the four fundamental forces of nature. It describes the interaction between quarks and gluons which make up hadrons such as protons and neutrons. In normal low temperature/density conditions quarks and gluons are permanently confined into hadrons. This means that a single quark cannot be isolated by a microscopic distance. However, QCD can also be used to calculate the properties of the quark gluon plasma that may have existed a couple of milliseconds after the big bang. During this time, due to the extreme temperatures, the quarks were unconfined.

At short distances or high temperatures the effective coupling is small and the problem can be solved using perturbation theory.

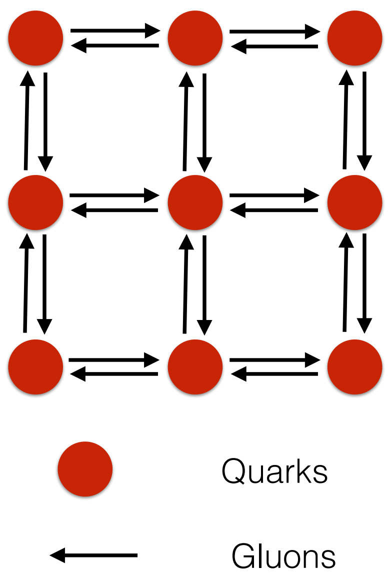


Figure 1: An Illistration of the lattice approximation, representing the lattice where the quarks/gluons are placed

However, as the distance is increased or the temperature decreased the interactions become too strong and perturbation theory is no longer an accurate approximation.

To solve larger systems a non-perturbative technique is necessary. The technique most often used is Lattice Quantum Chromodynamics (LQCD), which regularises QCD by introducing a space-time lattice with quarks occupying the lattice sites and gluon's being the links that join them together as shown in figure 1.

$$Dz = b \tag{1}$$

The computational challenge comes from repeatedly solving very large, sparse linear systems such as the one shown in equation 1. As the matrix (D) is extremely sparse (the number of non-zero entries only increases linearly with

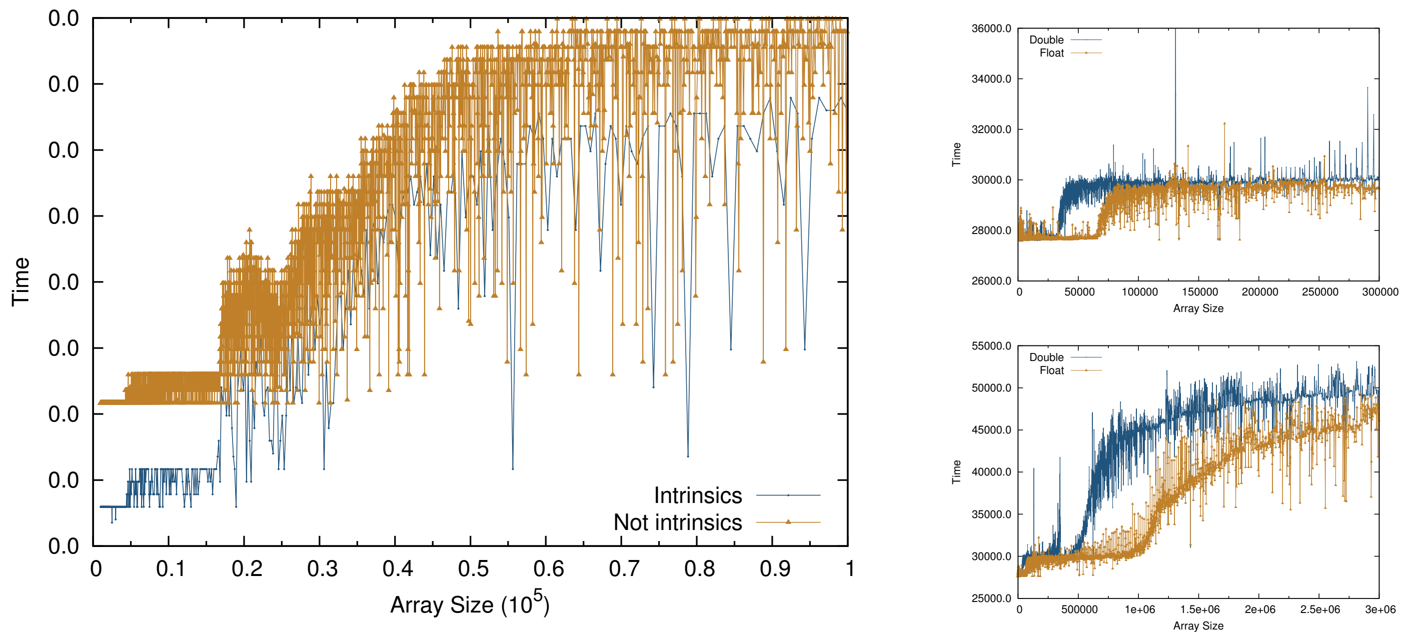


Figure 2: The time required to access array elements increases as the array size increases. The first figure shows how the code was improved by hand optimisation using Intel intrinsic. The second figures show how single/double precision operators effect the access time.

the matrix dimension(N), rather than N^2) it is most efficiently solved using a iterative solver. One such solver is the conjugate gradient (CG) method which can be used to find an approximate solution for z. However, even using these methods the simulations still involve a large amount of computational resources as the lattices can be as large as $144 \cdot 64^3$ lattice points or a value of N equal to 452,984,832. The dynamics and interactions of the quark fields and gauge fields are described using the equation below:

$$(\mathcal{D} + m)\psi(x) = \eta(x) \qquad (2)$$

ψ and η represent quark fields and the gluon's are represented by the Dirac operator($\mathcal{D}$) which is given by:

$$\mathcal{D} = \sum_{\mu=0}^{3} \gamma_\mu \otimes (\partial_\mu + A_\mu) \qquad (3)$$

In order to use these equations they must be discretised. However, this assumes that the lattice spacing is non zero and therefore the simulations must be extrapolated to the continuum limit. For this the simulations must be repeated at multiple sufficiently small lattice spacings.

What is the problem in repeatedly solving large matrices? The problem comes from the fact that large matrices use more of the limited memory available on the computer and as the matrix size increases the information will no longer fit in the fast cache memory and therefore has to be stored in main memory, which takes longer for the computer to access. On the Xeon Phi, it takes approximately 2 cycles to access the L1 cache, more than 20 cycles to access the L2 cache, and over 300 cycles to access the data stored in the GDDR memory. Therefore, ideally, all the data should be stored in the cache. The first step of the project was to calculate the size of the different caches in a typical architecture. A simple programme was written which calculated the time taken to multiply together two vectors:

$$\mathbf{a} = c\mathbf{a} + \mathbf{b} \qquad (4)$$

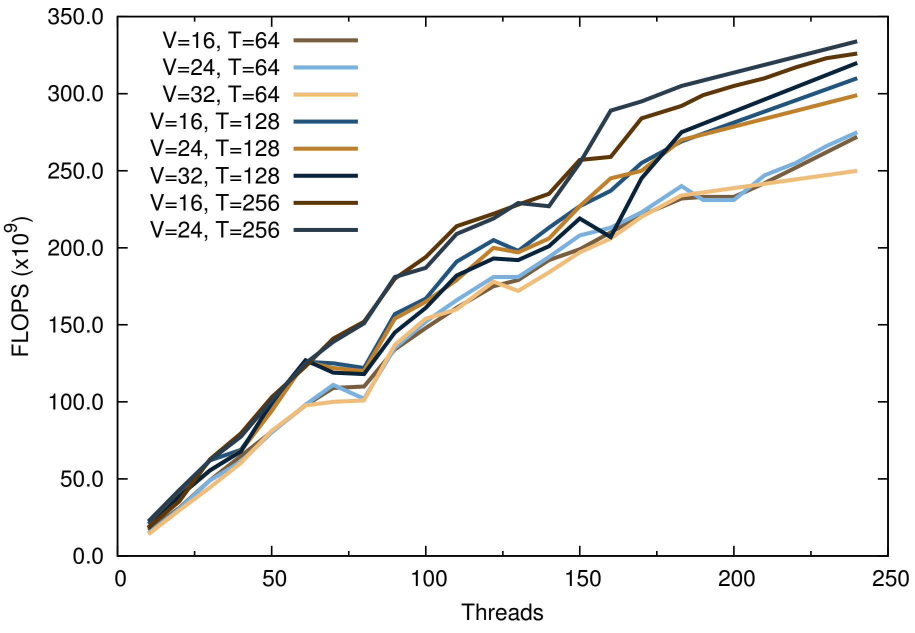
The results are shown in the figure above (there was too much noise to see the L1 cache). This gives an L2 cache of 32KB and an L3 cache size of 480KB, this is approximately the same as expected for my laptop.

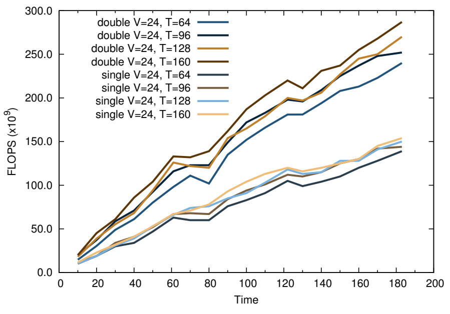
Now we want to calculate the number of FLOPS, this is the number of floating point operations per second and can be calculated using equation 5, where N is the number of floating point operations and t is the total time. A Floating point operation is any mathematical operation which uses floating point numbers. Therefore, the number of FLOPS is an important measure of a programmes speed and efficiency. For the simple vector multiplication and vector addition shown in (4) each element of the result vector requires two floating point operations. Therefore, the total number of floating point operations is two times the array size.

$$FLOPS = \frac{N}{t} \qquad (5)$$

The parts of the programme that take the most time are the memory accesses, therefore, ideally you want the smallest number of memory accesses per floating point operation. This should give a higher number of FLOPS.

The next step of the project is to attempt to improve the efficiency of the code. This is done using assembly level language, the so-called intel intrinsics. Whilst being more complex for you to programme, this is much simpler for the compiler to understand, because it mostly has a 1:1 mapping to assembler instructions. This means that you can create a programme which is optimised for a specific CPU, which should increase the efficiency of the programme. My hand optimised code was found to greatly increase the bandwidth by decreasing the time taken to do each floating point operation. This is shown in figure 2, for the intrinsics version the code was much more efficient especially at small array sizes. At larger





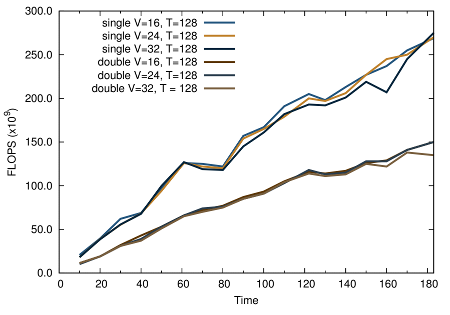


Figure 3: FLOPS for different LQCD systems as varied in space and time. This reaches a massive 300GFLOPS when run on the Xeon Phi.

array sizes, there is less of a difference because the floating point operations make up a smaller amount of the total time in comparison to the time taken to retrieve the information from memory.

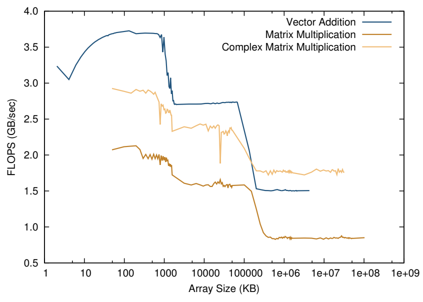


Figure 4: The change in FLOPS for different simple functions, as you can see the simple vector addition has the highest number of FLOPS

Next a more complicated function was used, a matrix multiplication and a complex matrix multiplication. This is to replicate the function used in the LQCD code.

$$\begin{bmatrix} D_{00} & D_{01} & D_{02} \\ D_{10} & D_{11} & D_{12} \\ D_{20} & D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} z_{00} \\ z_{10} \\ z_{20} \end{bmatrix} = \begin{bmatrix} b_{00} \\ b_{10} \\ b_{20} \end{bmatrix} \tag{6}$$

The FLOPS were calculated as the array size was increased and compared for each array size for the three different functions. As you can see the FLOPS are constant, until the array will no longer fit into one of the caches. Therefore ideally, you want to run the LQCD simulations just before the cache misses occur. Ideally at around 50,000KB, as this is large enough for the simulation to give realistic results but not so large that the memory wont fit in the L2 cache.

To test this, the LQCD package was run for different system sizes and for different times. This was done for different numbers of threads to measure the level of parallelisation. For high numbers of threads and large system sizes the programme reached a massive 300GFLop/s. This is because as the system size increases, the CPU can do a better job of prefetching the data, as the patterns of which data will be accessed next are more predictable.

This knowledge was then used to improve the LQCD package, by increasing the parallelisation of parts of the code using Simultaneous Multithreading (SMT). Here, we used pthreads rather than the more common openMP, in order to have maximal control over the threading.

In conclusion, this time at SoHPC has been a really useful learning experience, I have learnt a great deal about code optimisation and parallelisation and the best methods of optimising different codes for different CPU's.

PRACE SoHPCReferences

[1] Frommer, A., Kahl, K., Krieg, S., Leder, B., and Rottmann, M. Adaptive aggregation based domain decomposition multigrid for the lattice wilson dirac operator. *J.Sci.Comput.* **36** (2014),

[2] Gattringer, C., Lang, C., Quantum Chromodynamics on the Lattice *Springer*,

[3] DeGrand, T., DeTar, C., Lattice Methods for Quantum Chromodynamics *Springer*,

[4] Teukolsky, S., Press, W., Vetterling, W., Flannery, B., Numerical Recipes in C *Cambridge*,

PRACE SoHPCProject Title
Making Quarks Phli Further

PRACE SoHPCSite
Forschungszentrum, Juelich, Germany

PRACE SoHPCAuthors
Sarah Jenkins, [University of York,] United Kingdom

PRACE SoHPCMentor
Stefan Krieg, Forschungszentrum, Juelich, Germany

Sarah Jenkins

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

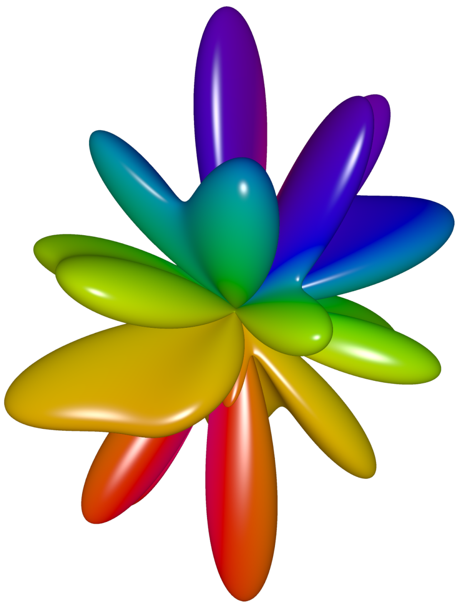
PRACE SoHPCAcknowledgement
I would like to thank my supervisor Stefan Krieg. I would also like to thank Eric Gregory for all of his help and advice, Ivo Kabadshow for co-ordinating the projectand everyone else here at the JSC. I would also like to thank Leon Kos and everyone at the BSC for all their help during the training week.

PRACE SoHPCProject ID
1515

Parallel FMM on a GPU a CUDA/C++ love story

Albert Garcia

The simulation of interactions in huge particle ensembles is a vital issue in current scientific research. The FMM is able to compute those Coulomb interactions with extraordinary speed and controlled precision. A key part of this method is its shifting operators, which usually exhibit $\mathcal{O}(p^4)$ complexity. There exist special rotation-based operators with $\mathcal{O}(p^3)$ complexity that will be used instead. However, they are still computationally expensive. In this work, we will parallelise those operators and deploy the implementation on a GPU to accelerate the FMM.



THE simulation of dynamical systems of particles, usually under the influence of physical properties, such as gravitational or electrostatic forces, is a crucial issue in scientific research. This problem is commonly referred to as the N-body problem. The main obstacle of that problem is the absence of an analytical solution when the number of bodies, N, is greater than three total bodies.

However, those simulations can be solved using an iterative numerical approach. Therefore, such methods compute the total force exerted on each particle and their potentials at discrete time steps, then compute the resulting velocities and update the positions of the particles accordingly.

A typical example is the simulation of a system of particles with electric point charges q_i. The force $\boldsymbol{F}_{ij}$ of a particle j with charge q_j acting on a particle i with charge q_i is defined by the following expression:

$$\boldsymbol{F}_{ij} = \frac{q_i q_j}{|\boldsymbol{r}_{ij}|^3}\boldsymbol{r}_{ij}\,,$$

where $\boldsymbol{r}_{ij}$ is the vector between particles i and j.

Given that, the total force $\boldsymbol{F}_i$ acting on each particle i can be expressed as the following summation:

$$\boldsymbol{F}_i = \sum_{j=1}^{N} \frac{q_i q_j}{|\boldsymbol{r}_{ij}|^3}\boldsymbol{r}_{ij} \quad (i \neq j)\,.$$

As we can observe, calculating the forces acting on a single particle has a computational complexity of $\mathcal{O}(N)$ since we have to compute all pairwise interactions of the current particle with the rest of the system. Therefore, a naïve algorithm for computing all forces $\boldsymbol{F}_i$ exhibits $\mathcal{O}(N^2)$ complexity.

The remainders of the simulation steps have a complexity of $\mathcal{O}(N)$ since computing the velocities by using the forces just needs to iterate once over each particle and the same applies for the position update step. In this regard, the quadratic complexity may be negligible for a small number of particles, but *interesting and useful simulations* often involve huge particle ensembles. Hence, the simulation will be considerably slowed down to a point in which it is non-viable to apply this kind of summation method. Fortunately, due to the increasing importance of N-body simulations for research purposes, fast summation methods have been developed throughout the latter years.

Arguably, the most remarkable one is the *Fast Multipole Method (FMM)*, introduced by Greengard and Rokhlin.[3]

This method is considered *one of the TOP10 algorithms of the past century*[2] together with other notable ones as the *Monte-Carlo method*, the *Quicksort algorithm* or the *Fast Fourier transform*. The mathematician Barry Cipra perfectly summarized[1] the functioning of the algorithm:

"This algorithm overcomes one of the biggest headaches of N-body simulations: the fact that accurate calculations of the motions of N particles interacting via gravitational or electrostatic forces (think stars in a galaxy, or atoms in a protein) would seem to require $\mathcal{O}(N^2)$ computations. The fast multipole algorithm gets

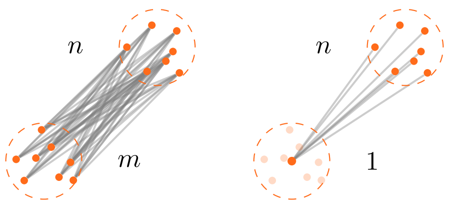
by with $\mathcal{O}(N)$ computations. It does so by using multipole expansions (net charge or mass, dipole moment, quadrupole, and so on) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever-larger groups as distances increase."

Theoretical Background

As we previously mentioned, the FMM is a fast summation method which is able to provide an approximate solution to the calculation of forces, potentials or energies within a given precision goal, namely ϵ. The method exhibits linear computational complexity, mainly thanks to its sophisticated algorithmic structure. In this section we will describe the core aspects of the method.

Particle Grouping

The main idea behind the FMM can be informally described as the following intuitive concept: the effect that particles, which are close to the observation point, also named target, have over that target particle is dominant compared to the effect produced by remote particles. It is important to remark that the contribution of those remote particles is not zero. Oherwise, we will be resorting to what is known as cut-off scheme. Those methods present a $\mathcal{O}(N)$ complexity too, but lack error control capabilities. That is why they are not used for accurate simulations.



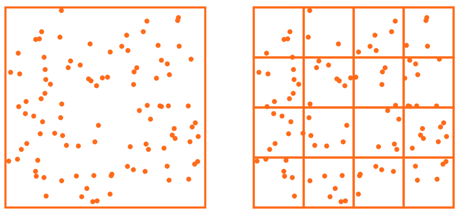
(a) $m \cdot n$ inter. **(b)** n interactions

Figure 1: Figure (a) shows the direct interactions of the particles of one cluster with all particles in the other. Figure (b) shows the interactions via a target pseudo-particle.

The FMM is based on the idea that a remote particle from a spatial cluster will have almost the same influence on the target particle as another one from the same cluster, given that the inter-cluster distance is large enough. This means that all particles in the remote cluster can be grouped together into a pseudo-particle. By doing this, the amount of interactions to be computed is reduced (see Figure 1).

Space Subdivision

In order to implement the spatial grouping idea, it is required to subdivide the simulation space to generate particle groups. The FMM uses a space decomposition scheme based on a recursive decomposition in cubic boxes. Firstly, the spatial simulation domain is enclosed in a three-dimensional box. Then, the cube is subdivided by planes parallel to its faces which pass through the box centroid. This decomposition technique generates eight different child cubes which are recursively subdivided. This hierarchy of cubes is arranged in a tree, namely an octree. The space is subdivided until a predefined depth d on that tree is reached. Figure 2 shows an example of this recursive subdivision.



(a) $d = 0$ **(b)** $d = 2$

Figure 2: Space subdivision using an octree, visualized in a 2D plane.

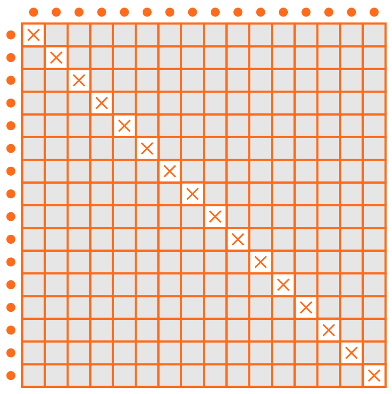
Interaction Scheme

As we previously mentioned, the FMM takes advantage of grouped remote particles to reduce the amount of interactions to be computed. This interaction scheme is the main characteristic of the FMM algorithm. Figure 3 shows the interaction matrix of a distribution of 16 particles using the direct summation method and the FMM.

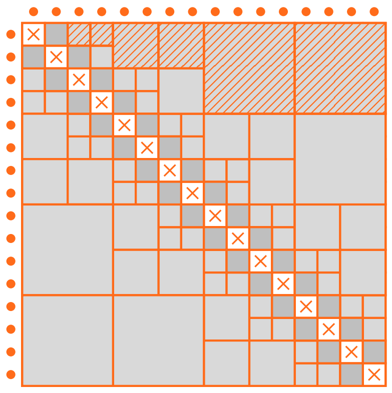
As we can observe, the direct summation scheme computes a total of $N(N-1)/2$ interactions. This is due to the interaction symmetry and the dropped out self-interactions. On the other hand, in the FMM scheme the source and target particles are grouped as the distance increases thus dropping the number of interactions significantly.

Multipole Expansions

In order to accurately depict the charge distribution of the created particle groups, the FMM resorts to multipole and local expansions. A group could be represented by reducing all the charges to a common point charge or monopole, but this is not precise enough to represent the charge distribution inside that cluster. Instead of doing that, the group is represented by a series of increasingly higher-order multipoles. The series is truncated to a certain order p, this parameter of the FMM controls the final accuracy and balances the computational load.



(a) Direct



(b) FMM

Figure 3: Figure (a) shows the interaction matrix of 16 particles (orange bullets) using a direct summation scheme. Figure (b) depicts the interaction matrix via the FMM scheme, the dark-grayed cells represent direct neighbors. Each square of a matrix represents a particle-particle or multipole-multipole interaction. Crossed out cells represent interactions from particles with themselves, which have to be left out.

The solution for the potential Φ of a particle P consists of two separate solutions: a multipole expansion with moments ω_{lm} for a remote part outside the sphere, and a Taylor-like expansion with moments μ_{lm} for the inside.

The multipole expansion is defined by the following theorem: suppose that k particles of charges q_j, with $j \in \{1, ..., k\}$ are located at the points $a_j = (a_j, \alpha_j, \beta_j)$, with $j \in \{1, ..., k\}$ and $|a_j| < \hat{a}$ inside the sphere. Then for any $P = (r, \theta, \phi) \in \boldsymbol{R}^3$ with $r > \hat{a}$, the potential is given by:

$$\Phi_P = \sum_{l=0}^{p} \sum_{m=-l}^{l} \omega_{lm}(q, a) M_{lm}(r) \,.$$

The Taylor-like expansion is defined by the following theorem: suppose that k particles of charges $q_j, j = 1, ..., k$ are located at the points $R_j = (r_j, \theta_j, \phi_j)$, with $j \in \{1, ..., k\}$ and $|r_j| > \hat{a}$ outside the sphere. Then for any $P = (a, \alpha, \beta) \in \boldsymbol{R}^3$ with $a < \hat{a}$, the potential is given by:

$$\Phi_P = \sum_{l=0}^{p} \sum_{m=-l}^{l} \mu_{lm}(q, r) O_{lm}(a) \,.$$

Workflow

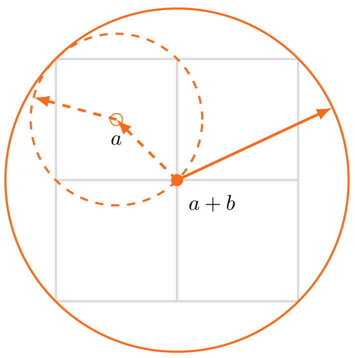
Once the core aspects of the FMM have been reviewed, we can put them all together to establish the general workflow of the algorithm. Given a certain separateness criterion ws, the multipole order p and the depth of the tree d, the FMM consists of the following steps:

- **Pass 1:** Expand particles into multipole moments ω_{lm} on the lowest leve and shift multipole moments ω_{lm} up the tree.
- **Pass 2:** Transform multipole moments ω_{lm} into local moments μ_{lm}
- **Pass 3:** Shift local moments μ_{lm} down the tree
- **Pass 4:** Compute far field contributions: potentials Φ_{FF}, forces $\boldsymbol{F}_{\text{FF}}$, and energy E_{FF}.
- **Pass 5:** Compute near field contributions: potentials Φ_{NF}, forces $\boldsymbol{F}_{\text{NF}}$, and energy E_{NF}.

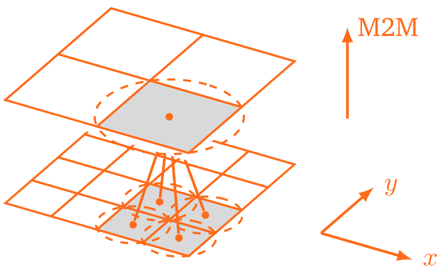
The actual phases of the algorithm are usually named *passes*. The first pass corresponds to the Particle-to-Multipole (P2M) and Multipole-to-Multipole (M2M) operators, while the second one is done via the Multipole-to-Local (M2L) operator and the third one with the Local-to-Local (L2L) operator. Those operators are responsible for shifting the multipole expansions up and down the tree levels, and also for converting remote multipole expansions to local ones at each level. This article focuses on these three operators: M2M, M2L, and L2L. More precisely, we will highlight the M2M operator, since the optimisations performed to it can be ported to the other ones.

Multipole-to-Multipole Operator

The Multipole-to-Multipole (M2M) operator is a vertical operator which shifts the multipole coefficients up to higher levels of the tree structure. Each box of the 3D tree has eight child boxes in the next lower level. The M2M operator sums up all the moments of the multipole expansions of the child boxes at the centre of the parent box. This operator is applied to each box on each level up to the root of the tree. By doing this, each box on every level holds a multipole expansion. Since this operator is applied in the first pass, it is also known as operator A.



(a) Analytical domain.



(b) Tree view.

Figure 4: (a) shows the analytical domain of the A operator on a 2D tree. The centers of a sample child box and the parent one are shown. (b) depicts the functioning of the M2M operator for a 2D system.

From a mathematical perspective, each child multipole expansion ω^i at the centre a_i of that child box i is shifted up to the centre $a + b$ of its parent box (see Figure 4). The moments $\omega^i_{jk}(a_i)$ of each child multipole expansion are shifted by the A operator to produce the moments $\omega^i_{lm}(a_i + b_i)$ of the parent via:

$$\omega^i_{lm}(a_i + b_i) = \sum_{j=0}^{l} \sum_{k=-j}^{j} A^{lm}_{jk}(b_i) \omega^i_{jk}(a_i) \,.$$

All the shifted moments of the eight child boxes are finally added up to conform the multipole expansion at the center of the parent box via:

$$\omega_{lm}(a + b) = \sum_{i=1}^{8} \omega^i_{lm}(a_i + b_i) \,.$$

The described operator, and M2L and L2L as well, have a computational complexity of $\mathcal{O}(p^4)$. This fact has a considerable impact when dealing with high-precision simulations in which the order of the multi-poles is usually high. Taking into account that the three passes take a considerable amount of the total execution time of a FMM step, it is critical to reduce the cost of the mathematical operators in order to achieve faster simulations.

Rotation-based Operators

A set of more efficient operators with $\mathcal{O}(p^3)$ computational complexity scaling were proposed by White and Head-Gordon[9] . The reduced complexity is achieved by rotating the expansions so that the translations or shifts are performed along the quantisation axis of the boxes (see Figure 5). This reduces the 3D problem to a 1D one.

Along the quantisation axis, the angles $\theta = 0$ and $\phi = 0$ so the operators take an especially simplified form[8] . In order to rotate the multipole or local expansions, Wigner rotation matrices are applied. Two successive rotations are performed: first one about the z-axis with a rotation angle ϕ, followed by another one about the new y-axis with a rotation angle θ. Those angles correspond to the polar and azimuthal angles of the shift vector $\boldsymbol{d}$, depicted as R in Figure 5.

The multipole moments of an expansion with respect to a coordinate system which has been rotated twice, first ϕ degrees about the z-axis and then θ about the y-axis, can be expressed as a linear combination of the moments with respect to the original coordinate system via:

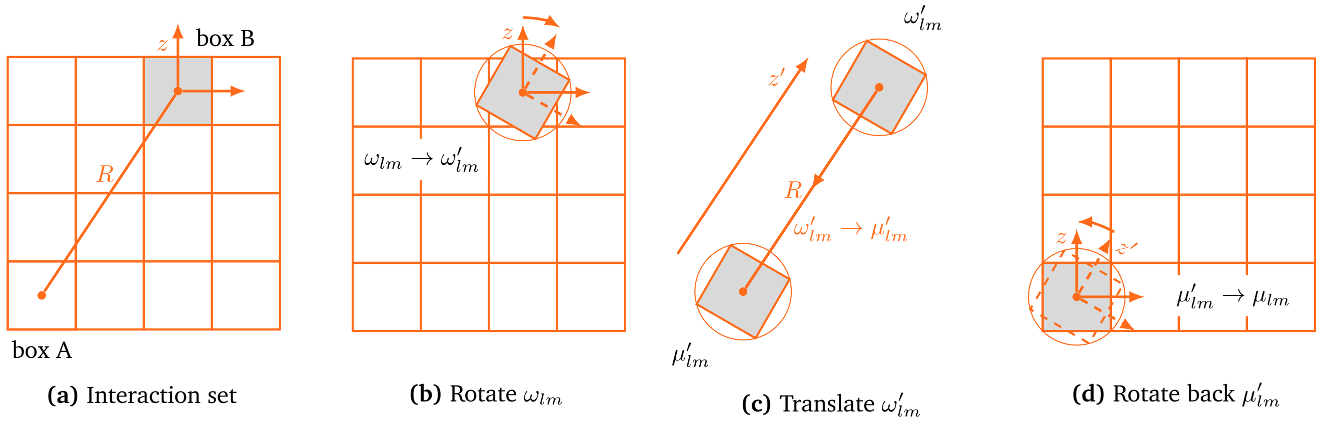

box B
box A
z
R
$\omega_{lm} \to \omega'_{lm}$
ω'_{lm}
z'
$\omega'_{lm} \to \mu'_{lm}$
μ'_{lm}
z'
$\mu'_{lm} \to \mu_{lm}$
(a) Interaction set
(b) Rotate ω_{lm}
(c) Translate ω'_{lm}
(d) Rotate back μ'_{lm}

Figure 5: Application of the rotation operator to align certain multipole expansion with another, step-by-step diagram. Figure reproduced from reference[5] .

$$c_l^{mk} = \frac{\sqrt{(l-k)!(l+k)!}}{\sqrt{(l-m)!(l+m)!}}\,,$$

$$\omega'_{lm} = \sum_{k=-l}^{l} c_l^{mk} d_l^{mk}(\theta) e^{ik\phi} \omega_{lk}\,.$$

The same applies for the local moments $\mu'_{l,m}$ of an expansion, which are expressed as a linear combination of the moments $\mu_{l,m}$ with respect to the original coordinate system, via:

$$\mu'_{lm} = \sum_{k=-l}^{l} c_l^{km} d_l^{mk}(\theta) e^{ik\phi} \mu_{lk}\,.$$

Our focus during this work was placed on these $\mathcal{O}(p^3)$ operators, and as we previously said, on the M2M one since it is the simplest and the optimisations applied to it can be ported to the other ones.

Goals

The goals of this project were the following ones:

- Explore the possibilities of using Graphics Processing Units (GPUs) and Compute Unified Device Architecture (CUDA) to develop a parallel version of the M2M $\mathcal{O}(p^3)$ operator for the FMM.
- Minimize the impact on the current project, and try to maintain a single codebase for both CPU and GPU operators.
- Make use of C++11 features if it is possible, and prepare the code for moving to C++14 eventually.

In order to accomplish those goals, we divided the project into two main tasks: set up an application layout based on abstraction layers to work with the GPU or the CPU indistinctively, and develop a set of CUDA-accelerated kernel functions for the rotation, operator, and rotation backwards steps for the M2M $\mathcal{O}(p^3)$ operator.

Abstraction Layers

From the application point of view, the implementation can be seen as a stack of abstraction layers, each one on top of another, having different responsibilities. By using this layered approach, the internal functionality of a layer can be changed without having to change the other ones. This design produces a flexible application, and it is implemented using C++templates in the different layers. Figure 6 shows the layered view of the current implementation.

As we can observe, the program is composed by five well distinguished layers: the algorithm layer, the data structures, the pool allocator, the allocator and the memory. The top layer contains the logic for the FMM itself, i.e., the implementations of all the passes. In our case, we keep the focus on the three aforementioned operators: M2M, M2L and L2L.

Furthermore, templates allow us to choose between the $\mathcal{O}(p^4)$ or $\mathcal{O}(p^3)$ operators. It is important to remark that, thanks to the usage of an inner ab-

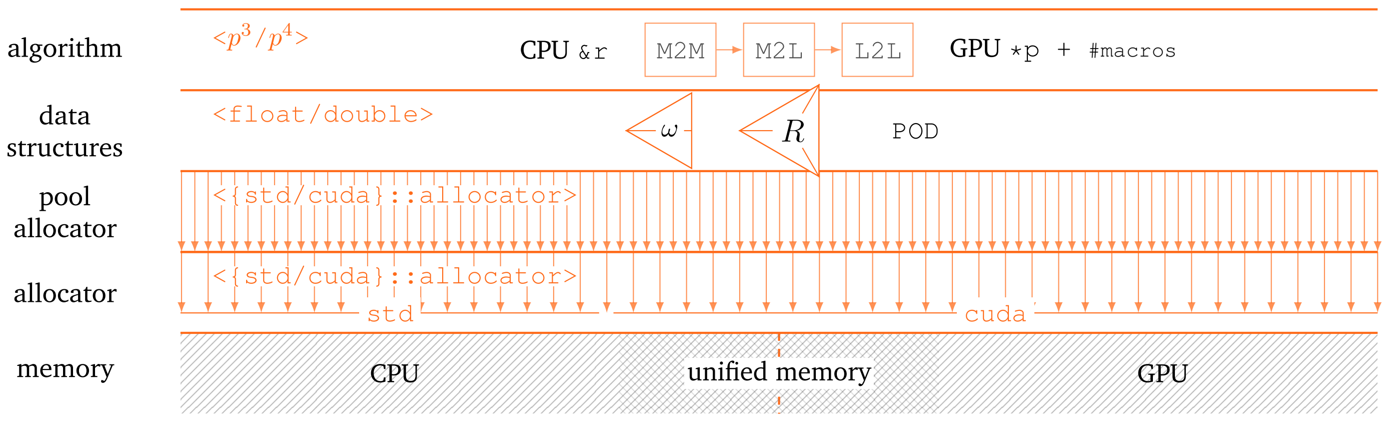


Figure 6: Layout of the application with five abstraction layers and a single codebase.

straction layer to convert references to pointers, and also a set of preprocessor macros to determine loop ranges depending on whether we want to execute the operator on the GPU or on the Central Processing Unit (CPU), a single merged codebase exists for both implementations.

Those implementations need data structures to store the information which is being processed. In this regard, the algorithm layer resorts to the data structures one. This layer contains the data types needed for the algorithm, e.g., coefficient matrices (ω or μ), rotation matrices (R), or other simple data structures, together with their internal logic. Thanks to the templated design, we are able to choose the precision of the underlying data types independently.

At last, the data structures need to allocate memory for storing their information. The allocator layer is responsible for this action. The data structures delegate the memory allocation to an allocator which will perform the corresponding calls for allocating and also deallocating memory. Thanks to the templates, this action can be performed by any allocator.

Between the data structures and the actual allocator, an intermediate layer exists: the pool allocator. This middle level receives the allocation calls issued by the data structures and serves parts of chunks of memory which have been previously allocated by actual calls to the allocator layer. This layer increases the performance of the allocation since less calls have to be performed, and also allows us to overcome possible allocation calls limits when using fine grained data structures.

Apart from this layout, we developed a custom allocator for the GPU which is able to allocate unified memory thanks to CUDA managed allocation functions. This custom allocator is then plugged in as a template parameter to our allocator layer. In this way, we can allocate space in the GPU without adding additional logic to the data structures or the algorithms.

CUDA Implementation

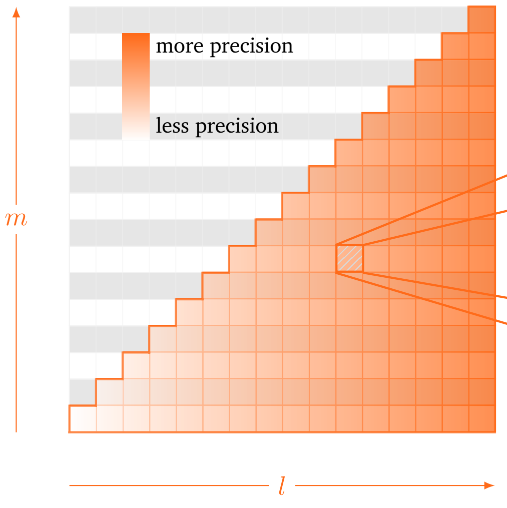
Once this layout was deployed, we were able to start running the operators on the GPU. For that purpose, we implemented three separated CUDA kernels: rotation, M2M operator, and rotation backwards. Those kernels are executed sequentially by a GPU stream so the full M2M can be executed on a GPU with a simple function call.

The first step consisted of exposing enough parallelism for the GPU, i.e., launching enough threads with enough work to keep the graphics processor busy and get performance out of it by hiding parallelism overhead and latencies. In order to do that, we analysed the main data structure handled by the three aforementioned steps: the coefficient matrix (see Figure 7).

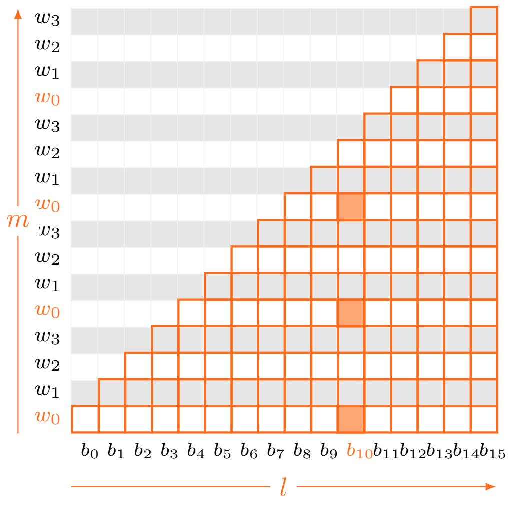
After doing that, we decided a parallelisation schema in which we will assign one block to each column of each coefficient matrix. Then our grid has a size of poles $\times$ boxes. The grid x dimension corresponds to the number of poles that the multipole requires, i.e., l-axis in the coefficient matrix. The grid y dimension corresponds to the number of boxes or coefficient matrices that we have to process. In addition, each block will have a 32×4 threads configuration, so each one is composed of four warps. Those warps will be assigned to individual elements of a column of the coefficient matrix depending on their corresponding block. Figure 7 shows an example of block and warp work distribution.

Once we have deployed an appropriate way to expose parallelism for the GPU, we implemented the kernels for the rotation, M2M operator, and rotation backwards. We took advantage of a set of CUDA basic techniques and advanced optimisation tricks, including external libraries, and architecture specific operations. The features that we implemented are:

- Flexible and scalable grid-strided loops[4] so that we can deal with any problem size in an efficient


more precision
less precision
m
l
Multipole moment: ω_{lm}
$\omega_{lm} \in \mathbb{C}$
$a_{lm} + ib_{lm}$
or local moment: μ_{lm}
$\mu_{lm} \in \mathbb{C}$

(a) Coefficient matrix


w_3 w_2 w_1 w_0
m
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}
l

(b) GPU distribution

Figure 7: Figure (a) shows a representation of the coefficient matrix datatype with an arbitrary number of 15 poles. As the coefficient matrix grows, the precision increases, so a higher number of poles leads to more accurate results. Both axes are in the range $[0, p]$, this produces a coefficient matrix of $(p + 1)(p + 2)/2$ coefficients if only the upper part is considered. The coefficients, which are represented as square blocks in the picture, are multipole or local moments depending on the type of the expansion. Each one of them represents a complex number. Figure (b) shows the block and warp distribution strategy to expose parallelism, also an example of work assigned to warp zero from block ten is shown.

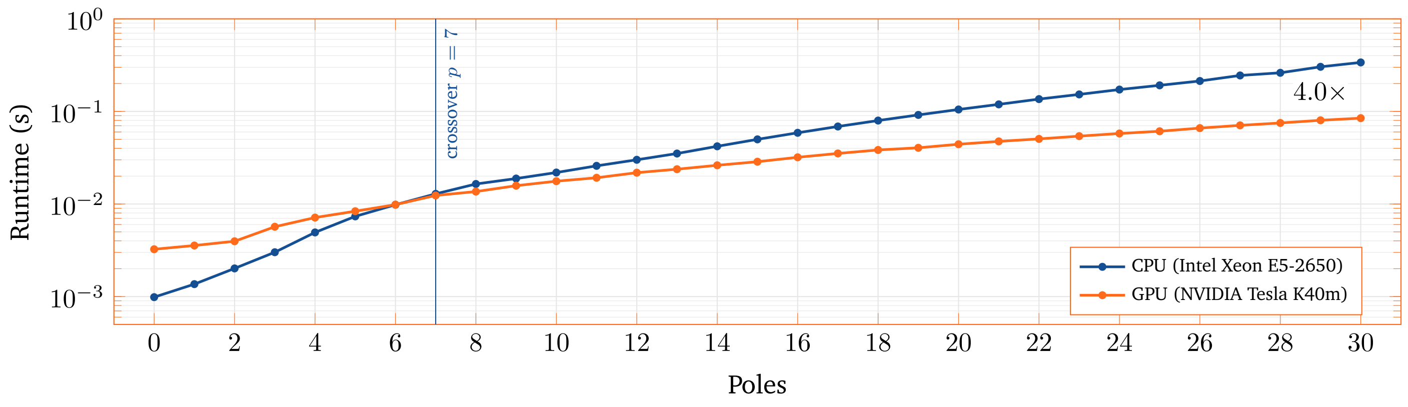


Figure 8: Timing results of the experimentation carried out to determine the performance of the CUDA-accelerated implementation. The results correspond to a full M2M pass over the lowest level of a tree with $d = 4$ using double precision.

manner.

- Fully coalesced accesses to the kernel data structures.
- Fast warp reductions[6] using CUDA Unbound (CUB)[7] .
- Launch bounds to help the compiler optimize the kernels if we know the block size in advance.
- Precomputed factors to save global memory round trips.

Results

Figure 8 shows the results of our experimentation with the CUDA-optimized M2M operator against a thoroughly optimized CPU version. The experimentation consisted of a set of full M2M passes with varying number of poles. The experiments were run on the *JUHYDRA* cluster (see Figure 9) whose specifications are the following ones:

- MEGWARE MiriQuid GPU-Server (2-Socket ES-2600)
- 2× Intel Xeon E5-2600, 8 cores, 2.0 GHz, 20 MiB cache
- 8× 8 GiB DDR3, 1600 MHz, ECC
- 2× 1 TiB SATA HDD, 7200 rpm
- 4× PCI-Express Gen.3.0 ×16 slots for dual-slot GPUs
- 2× NVIDIA Tesla K20Xm (K20Xm)
- 2× NVIDIA Tesla K40m (K40m)

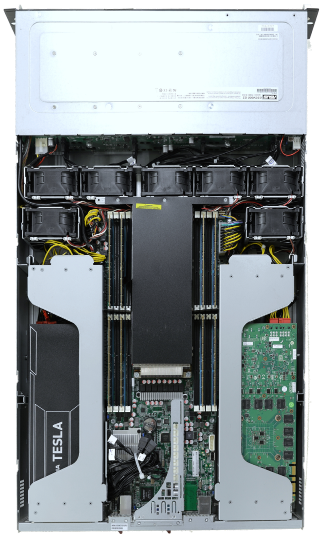


Figure 9: Top view of the JUHYDRA cluster.

The GPU version was executed on a K40m. As we can observe, the sequential version is faster when using up to 6 poles. After 7 poles, the CUDA-accelerated M2M starts getting faster than the optimised CPU one. In the end, for 30 poles, which was the biggest number of poles tested, an approximate speedup of 4.0× is achieved, compared to the single threaded CPU version.

Conclusion

In this work we have shown how to integrate GPU and CPU code into a single codebase using a flexible design, based on a set of abstraction layers to decouple responsibilities. We also have shown how the M2M operator can be accelerated using CUDA features. There is still room for improvement and the final speedup is not too high. However, all the optimisations can be ported to the M2L operator which exhibits a significantly higher computational load so a considerable gain is expected by parallelising that operator.

References

[1] B.A. Cipra. The best of the 20th century: editors name top 10 algorithms. *SIAM news*, 33(4):1–2, 2000.

[2] J. Dongarra and F. Sullivan. Guest editors introduction to the top 10 algorithms. *Computing in Science Engineering*, 2(1):22–23, Jan 2000.

[3] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[4] M. Harris. Cuda pro tip: Write flexible kernels with grid-stride loops.

[5] I. Kabadshow. *Periodic boundary conditions and the error-controlled fast multipole method*, volume 11. Forschungszentrum Jülich, 2012.

[6] J. Luitjens. Faster parallel reductions on kepler.

[7] D. Merrill. Cub, 2013. GPU Technology Conference.

[8] S. Pfreundschuh. Implementation of a rotation operator for the fast multipole method, 2014.

[9] C.A. White and M. Head-Gordon. Rotating around the quartic angular momentum barrier in fast multipole method calculations. *The Journal of Chemical Physics*, 105(12):5061–5067, 1996.

PRACE SoHPCProject Title
A Fast Multipole toolbox for a GPU cluster

PRACE SoHPCSite
Juelich Supercomputing Centre, Germany

PRACE SoHPCAuthors
Albert Garcia, University of Alicante, Spain

PRACE SoHPCMentor
Andreas Beckmann, JSC, Germany

Albert Garcia

PRACE SoHPCContact
Ivo, Kabadshow, JSC
Phone: +49 2461 61-8714
E-mail: i.kabadshow@fz-juelich.de

PRACE SoHPCSoftware applied
C++, CUDA, GCC, Boost, CLANG

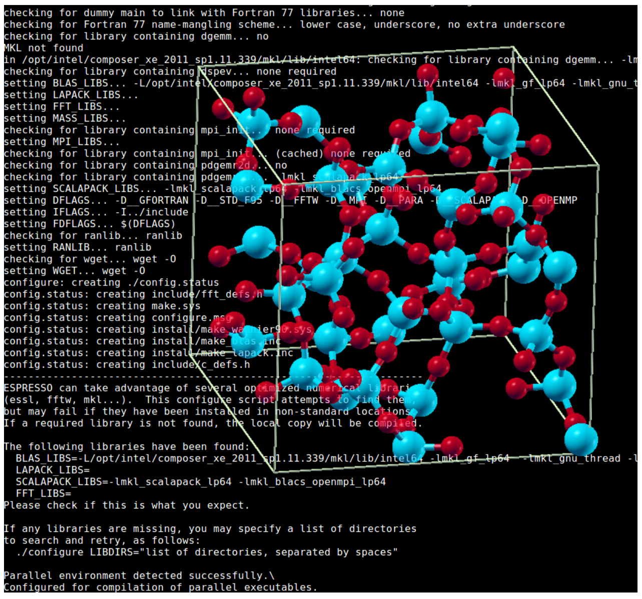
PRACE SoHPCAcknowledgement
I would like to thank the Juelich Supercomputing Center for their hospitality and for letting me use their resources during my stay. I would also like to express my gratitude to my mentors, Ivo and Andreas for the hours of coding, writing and learning. In addition, I would like to thank Jiri Kraus from NVIDIA for his support about GPU computing. Honorable mention awarded to Willi Homberg for keeping JUHYDRA alive despite our furious efforts to crash it. At last, thanks to PRACE and the SoHPC organisers for their effort which made this great opportunity possible.

PRACE SoHPCProject ID
1516

Bringing Hybrid Architecture’s power for Atomistic Simulations

Jan Hreha

Computer simulations of atoms and their electronic structure from first principles of Quantum Mechanic provide us unique view into the nano-scale world. However they demand high computational resources for precise results. We are seeking an answer whether utilising existing tools on new trendy GPU and Acceleration cards architectures brings desired speedup.



```
checking for dummy main to link with Fortran 77 libraries... none
checking for Fortran 77 name-mangling scheme... lower case, underscore, no extra underscore
checking for library containing dgemm... no
MKL not found
in /opt/intel/composer_xe_2011_sp1.11.339/mkl/lib/intel64: checking for library containing dgemm... -lm
checking for library containing dspev... none required
setting BLAS_LIBS... -L/opt/intel/composer_xe_2011_sp1.11.339/mkl/lib/intel64 -lmkl_gf_lp64 -lmkl_gnu_t
setting LAPACK_LIBS...
setting FFT_LIBS...
setting MASS_LIBS...
checking for library containing mpi_init... none required
setting MPI_LIBS...
checking for library containing mpi_init... (cached) none required
checking for library containing pdgemr2d... no
checking for library containing pdgemr2d... -lmkl_scalapack_lp64
setting SCALAPACK_LIBS... -lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64
setting DFLAGS... -D__GFORTRAN -D__STD_F95 -D__FFTW -D__MPI -D__PARA -D__SCALAPACK -D__OPENMP
setting IFLAGS... -I../include
setting FDFLAGS... $(DFLAGS)
checking for ranlib... ranlib
setting RANLIB... ranlib
checking for wget... wget -O
setting WGET... wget -O
configure: creating ./config.status
config.status: creating include/fft_defs.h
config.status: creating make.sys
config.status: creating configure.msg
config.status: creating install/make_wannier90.sys
config.status: creating install/make_blas.inc
config.status: creating install/make_lapack.inc
config.status: creating include/c_defs.h
--------------------------------------------------------------------
ESPRESSO can take advantage of several optimized numerical libraries
(essl, fftw, mkl...).  This configure script attempts to find them,
but may fail if they have been installed in non-standard locations.
If a required library is not found, the local copy will be compiled.

The following libraries have been found:
  BLAS_LIBS=-L/opt/intel/composer_xe_2011_sp1.11.339/mkl/lib/intel64 -lmkl_gf_lp64  -lmkl_gnu_thread -l
  LAPACK_LIBS=
  SCALAPACK_LIBS=-lmkl_scalapack_lp64 -lmkl_blacs_openmpi_lp64
  FFT_LIBS=
Please check if this is what you expect.

If any libraries are missing, you may specify a list of directories
to search and retry, as follows:
  ./configure LIBDIRS="list of directories, separated by spaces"

Parallel environment detected successfully.\
Configured for compilation of parallel executables.
```

Scientific motivation

The microscopic universe of atoms and molecules differs as much from our daily macroscopic experience as ocean’s depths from the shore. Laws of Physics apply to the elementary particles contrary to our intuition. It is difficult to find the right language when describing microscopic phenomena. Luckily there is mathematical language of Quantum Mechanics in which we can formulate statements and predictions. For many particle phenomena, like the properties of material consisting of enormous number of particles we utilise simplification models such as DFT (Density Functional Theory) to be able to calculate anything. Formulae and calculations are so complicated, that we have to employ computers to solve all but trivial cases.

Recent algorithms and tools provide remarkable results in fields such as Material Science, where we can compose materials with mechanical and electric properties tailored for specific use, and even model biochemical processes in cells for medicine and pharmaceutical industry.

Use those big machines

However this is possible only at high cost of enormous computational resources (supercomputers). Hence adopting existing proved software tools for new massively parallel architectures is promising speedup in both time and hardware expenses.

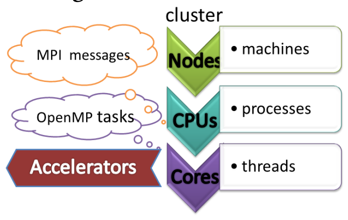
Parallel world

Since the year 2000 there was not much speedup in frequency with which the processors perform simple numeric operations, but we still follow Moore’s law of doubling their computational capacity every two years. That progress is achieved by doing sophisticated operations and calculating them simultaneously.

Hybrid hardware

Modern computers and even smartphones utilise heterogeneous processing units - a small number of general purpose processing (CPU) cores with specialised chips for accelerating different specific kinds of calculations, e.g. graphics (GPUs).

Building High Performance Computing infrastructure leads into grouping and interconnecting computing nodes which themselves contain multiple processors, each with many cores. Such clusters may contain from hundreds up to millions of cores. In the latter case, reasonable distribution of work and associated data is a challenge even for technology leaders. Independent program processes communicate via MPI messages. OpenMP governs cooperation between computation threads when processing data in shared memory.



New programming paradigms

The power of the accelerators lays in their ability to perform short similar routines in parallel over bigger chunks of data divided into smaller pieces.

GPUs run so called kernel programs written in special language (mainly CUDA), while Intel Phi cards employ 60 cores able to accelerate standard computer code Math Kernel Library routines.

Therefore, to utilise the computing capacity of hybrid computation environment one must learn completely new ways of programming.

The easier way is to decorate your serial code with Pragma Directives. These serve as suggestions for the compiler how the code parts may run in parallel if possible. Then, in runtime, frameworks such as OpenACC or OPENMP administer parallelism according to host architecture. The harder way leading to better performance is programming your own parallel functions and handling all the cooperation manually. At least for standard tasks you can take advantage of linking many parallel libraries already tuned by experts.

Application

We were given an opportunity to try to build and run accelerated simulations on different architectures. We have become acquainted with six PRACE partner's clusters in Hungary and one nVidia cluster in Italy. Thus we have worked with both AMD Opteron and Intel Xeon processors of different generations. In addition we have tried several type of accelerators: dual GPU nVidia Tesla M2070 with Fermi architecture, dual GPU nVidia Tesla GPU K20x with Kepler microarchitectre and even the newest nVidia GPU model K80 (Kepler). Moreover we have also tested Intel Xeon Phi x100 acceleration cards with 60 x86 cores.

Building PRACE partner's simulation software Quantum Espresso[1] means compiling the source code and linking it with proper libraries. We have gotten familiar with the whole process of setting up right software environment on given platforms. We got some experience with GNU and Intel Compilers, various Open/Intel MPI versions compiled with different compilers. We have even succeeded in installing and setting up the newest Intel Composer suite.

The most problematic part was linking Quantum Espresso to ScaLAPACK (ScaLAPACK stands for the whole family of parallel algebraic solvers) libraries. The built-in configuration script was insufficient, vendors documentation poor and online tutorials may work only on author's configuration.

Results

As a benchmark we have used Structural optimisation for 109 atoms sample of crystalline silica from PRACE partners benchmark package.[2]

In the following section we present results calculated on cluster Debrecen2 (HP SL250s) with two 8-cores Xeon E5-2650 v2 CPUs per node accompanied with dual nVidia K20 GPUs.

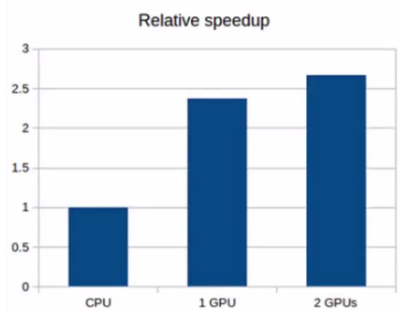


Figure 1: Speedup comparison: CPUs only versus one and two GPUs respectively.

As the main result we may present GPU caused speedup of factor nearly 2.4. When using 2 GPUs in dual mode, the speedup factor is over 2.6.

Then we focused on scaling the run over more resources. The time of calculation on fully loaded accelerated nodes has saturated at 48 cores and 6 GPUs. This scale is too small for first class supercomputers.

Therefore we have decided to examine scaling over number of GPUs, while using only necessary CPU resources (two MPI tasks for two GPUs per node). We have observed exponential decay of speedup i.e. increasing resources 8 times leads into only doubling of performance.

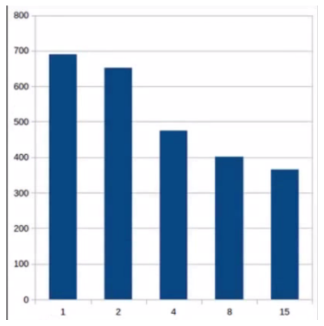


Figure 2: Computation time scaling by number of GPU nodes.

To conclude our work we can say, that new hybrid architectures bring remarkable speedup, but the scaling of the performance is suboptimal. Moreover building the software is often a complicated process with high time demands.

References

[1] Paolo Giannozzi, Stefano Baroni et. al.: QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials, Journal of Physics: Condensed Matter (2009), http://www.quantum-espresso.org

[2] Quantum ESPRESSO Benchmark Suite http://tinyurl.com/QEbenchmark

PRACE SoHPCProject Title

Visualisation and GPU Performance Analysis of PRACE Material Science Community Code: Quantum Espresso

PRACE SoHPCSite

National Information Infrastructure Development Institute (NIIF), Budapest, Hungary

PRACE SoHPCAuthors

Jan Hreha, Comenius University in Bratislava, Slovakia

PRACE SoHPCMentors

Gábor Roczei, NIIF, Hungary
Tamas Hornos, NIIF, Hungary

Jan Hreha photo

PRACE SoHPCSoftware applied

Quantum Espresso, Intel Composer suite, SLURM, Vesta

PRACE SoHPCMore Information

PRACE Summer of HPC
SoHPC blogs

PRACE SoHPCAcknowledgement

I would like to thank PRACE and NIIF for the provided HPC resources and to our mentors for friendly and skilled guidance.

PRACE SoHPCProject ID

1517

SIESTA, QE and HPC

Juraj Mavračić

SIESTA and Quantum Espresso mean a lot of hard work for a supercomputer. This packages are used to simulate matter and materials, which is one of the most resource demanding applications for HPC.

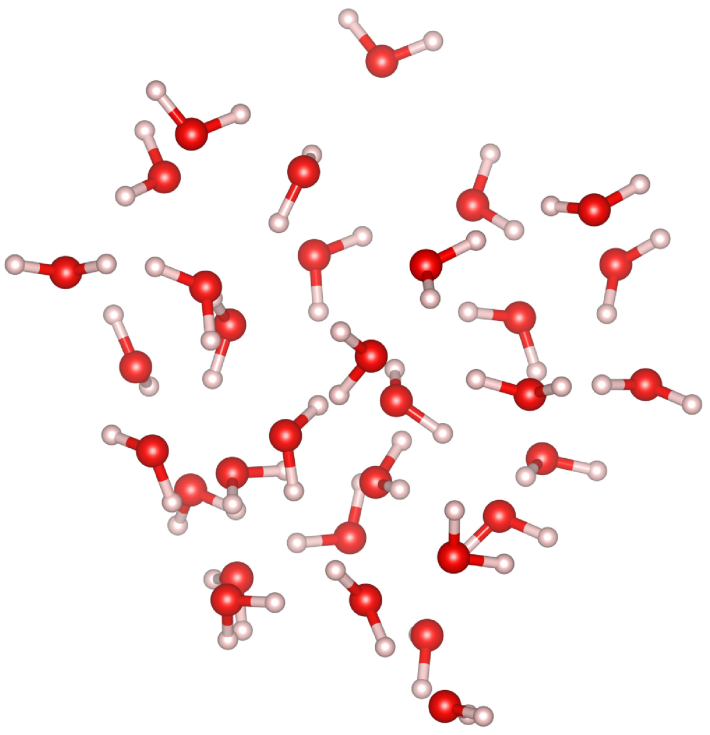


Figure 1: 32 water molecules in fluid state, simulated with SIESTA

Programs like Quantum Espresso[2] and SIESTA[1] make it possible to perform *ab-initio* calculations of electronic structure and molecular dynamics. Ab-initio means that structures are calculated using first principles of theoretical physics purely, without any empirical or experimental parameters needed during the calculation.
The results of such calculations are the most sophisticated and advanced models of matter currently available, thus enabling us to calculate various physical and chemical properties of materials and molecules, including dynamical properties like vibration and reaction pathways.
During this project various requirements for successfully running of this applications have been studied in detail. Unlike widely used desktop applications, which can be run without specific knowledge on programming and computer architecture, HPC applications are mostly much more complicated on usage. In order to successfully use HPC applications a lot of preparation steps need to be taken. The most important step is probably the compilation of the software on chosen HPC machines. Since performance is the most important factor to consider in HPC, a machine specific software configuration is the only way to guarantee best possible performance. This step is also one of the most time consuming parts during preparation of calculations.
In this project I focused on various aspects of working with supercomputers. The compilation of Quantum Espresso and SIESTA is only one of them. A lot of the time was spent on self-study and getting familiar with most important topics needed for autonomous work on a supercomputer. In addition, various Visualisation techniques have been studied.

Resources and workflow

During the project 4 of the 6 Hungarian supercomputers have been used, namely Budapest2, Szeged, Budapest and Debrecen2. For all of the HPC machines, the SLURM scheduler is used for submitting jobs. Once parameters for calculations are set right, the submitting of jobs is quite easy. For most of the HPC machines we were given reservations so we could experiment straight forward without waiting for our calculations to start.
For Visualisation, three programs have been used extensively: VESTA, XCrysDen and VMD. This are the most common tools in molecular modeling. They are capable of reading many different file formats. However, additional tools have been used for post-processing during Visualisation of SIESTA output. A lot of specific tools can be found for free on the internet. Since I was supposed to run jobs with different supercomputers and different software I even tested the trial version of Deneb by Atelgraphics. This is a

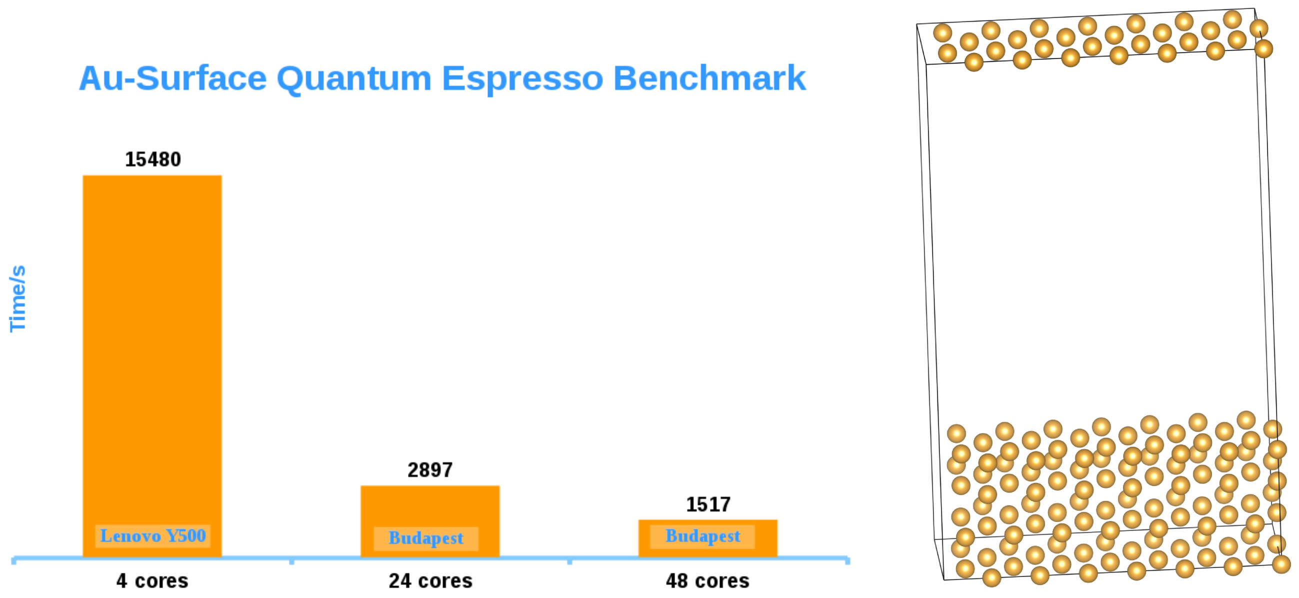


Figure 2: Au-Surface standard DEISA benchmark for QE. The results are shown on the lhs, on rhs the corresponding simulation cell is visualised. The free space in the middle is needed in order to prevent one surface of Au-atoms to interact with the other surface.

software package designed to manage this kind of work easily. All of the used programs run on Linux systems.

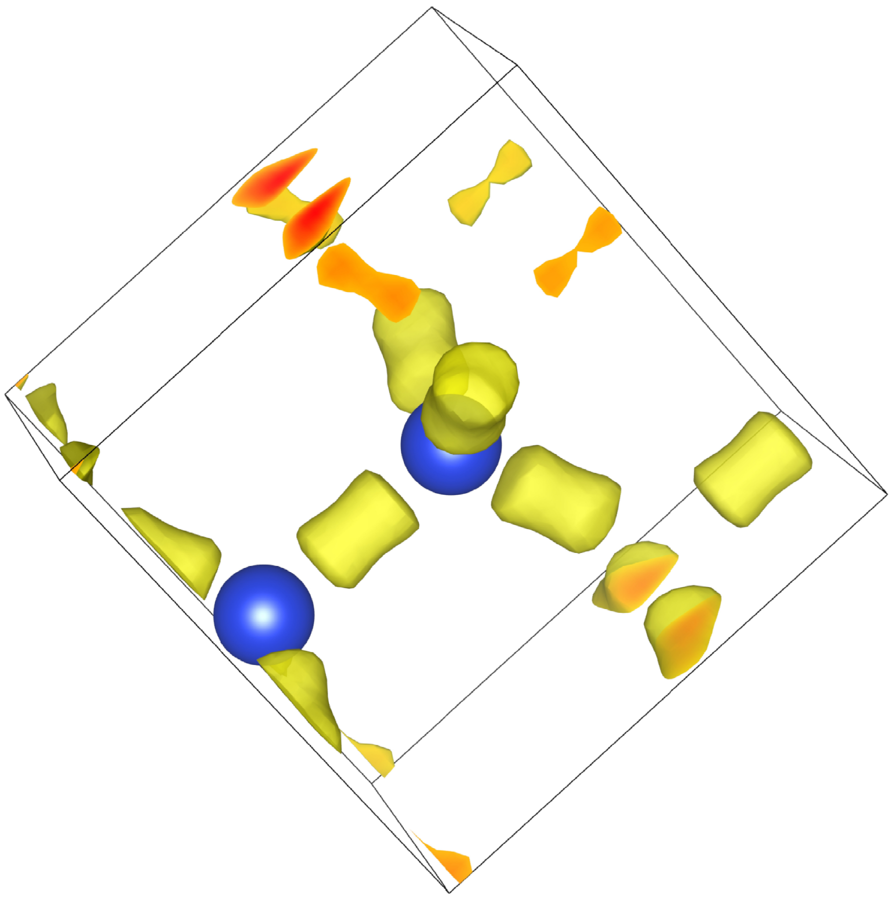


Figure 3: Charge density in a Si crystal. Electrons are located mostly between the Si atoms. Calculated with SIESTA.

Linux is probably the most common operating system for professional scientific usage. Therefore, I spent a lot of time on proper configuration of my OpenSUSE 13.2 distribution. I used my laptop mostly for accessing the supercomputers but also for visualising some of the output calculated on HPC machines. The most difficult part was the proper configuration of Nvidia drivers, which are still quite buggy for Linux systems. The configuration of SLI and glx was very time consuming. In addition to my local laptop, HPC machines have also been used for Visualisation, on specialised Visualisation nodes. To use this resources VirtualGL and VNCViewer have been used. Before the start of the actual calculations with SIESTA and Quantum Espresso I did some MPI testing to see if the SLURM scheduler directives and the supercomputers work properly and to get accustomed to it. For that purpose I created a small Fortran based MPI program containing a Monte-Carlo algorithm.

In HPC, performance significantly depends on the application which is run, on the code itself, but also on problem sizes and data distribution. Usually, there is a constant battle between calculation and communication. In many cases it is cheaper to perform calculations than to transfer data between different processors.

After getting familiar with SLURM and MPI it was time to learn about the specific SIESTA and Quantum Espresso input format. The directives are quite similar in many of the available ab-initio codes for materials modeling, but often there are details which shouldn't be left out for the calculations to run as supposed. Sample calculations and examples are probably your best friend when starting to work with a new code of that sort. The compilation of Quantum Espresso and SIESTA turned out to be the most difficult step in the project. For a successful build various libraries have to be linked during the compilation process. This becomes quite confusing very fast, specially if the various configuration options, paths to libraries and hardware specifications are not known or understood exactly. Mostly due to curiosity I have installed Quantum Espresso in both serial and parallel version on my Lenovo Y500 laptop for comparison. I performed some testing using different linear algebra packages, trying out Lapack, Blas, Atlas and OpenBlas, which has also an OpenMP version. The differences were not significant when tested with SIESTA. It seems that the standard Blas libraries are already highly optimised on my system.

Ab-initio means that structures are calculated using first principles of theoretical physics purely, without any empirical or experimental parameters needed during the calculation. The results of such calculations are the most sophisticated and advanced models of matter currently available.

Results and Discussion

For CPU performance analysis of Quantum Espresso, a standard QE benchmark has been introduced and run on the Budapest supercomputer. The DEISA standard medium sized benchmark has been used. The sample input is an Au-surface (Figure 2). Some minor changes to the input parameters have been made in order for the calculation to start successfully. The results are shown in Figure 2.

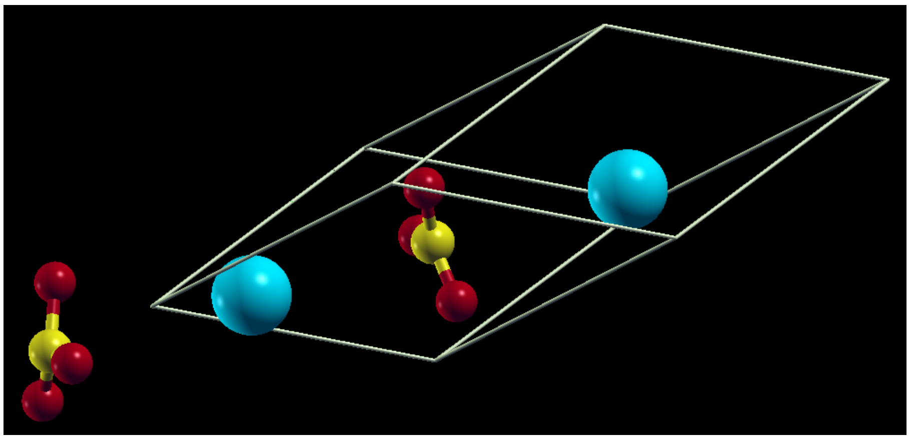


Figure 4: One asymmetrical repetitive unit of a $MgCO_3$ crystal. Molecular dynamics has been calculated with SIESTA and a video is provided separately.

A few sample calculations have been performed using SIESTA, for the purpose of trying out functionality. The charge density within a Si crystal has been visualised (Figure 3). It gives information about the positions of electrons within the structure. As expected, electrons are located mostly between neighbouring Si atoms. Each Si atom is tetrahedrally coordinated. A molecular dynamics run has been performed with $MgCO_3$ (Figure 4), with a thermostat on the basis of the Nosé–Hoover algorithm and a temperature of 600 K. A short video of the results is provided separately. Finally, a set of 32 water molecules (Figure 1) and a carbon-nanoscroll (Figure 5) have been simulated.

In addition to the SIESTA and QE calculations I did test the scalability of my Fortran MPI Monte-Carlo program on the Szeged supercomputer with a maximum of 96 cores. In that range it scaled quite well.

I can conclude that most of the difficulties during my project did arise as a consequence of insufficient documentation on available open source software. I hope that this will change in future, specially after the rise of clever and elegant documentation tools such as Sphinx.

The PRACE Summer of HPC was an excellent opportunity for me to gain additional knowledge in the field of high performance computing. I am looking forward to using this knowledge in my future work.

Abstract

The freely available ab-initio codes SIESTA and Quantum Espresso have been used for performing sample molecular modelling calculations and simple performance testing on CPU. As a result, various molecular Visualisations are presented including a molecular dynamics Visualisation of $MgCO_3$.

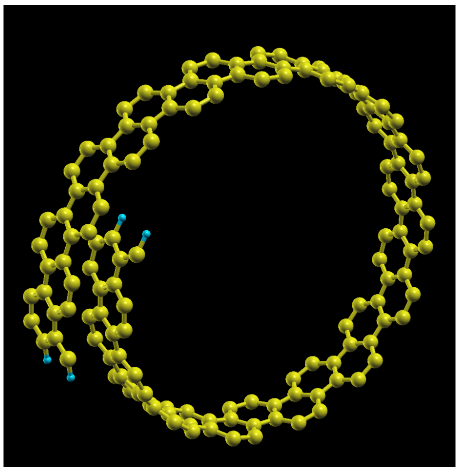


Figure 5: Carbon nanoscroll, calculated with SIESTA

References

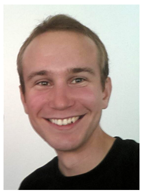
[1] J. M. Soler et al. (2002). The SIESTA method for ab-initio order-N materials simulation *J. Phys.: Condens. Matt.*, 14:2745-2779.

[2] P. Giannozzi, et al. (2009). QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials *J. Phys.: Condens. Matt.*, 21:395502.

PRACE SoHPCProject Title
Visualisation and CPU Performance Analysis of PRACE Material Science Community Code: Siesta

PRACE SoHPCSite
NIIF, Hungary

PRACE SoHPCAuthors
Juraj Mavračić, Austria


Juraj Mavračić

PRACE SoHPCMentor
Gabor Roczei, NIIF, Hungary
Tamas Hornos, NIIF, Hungary

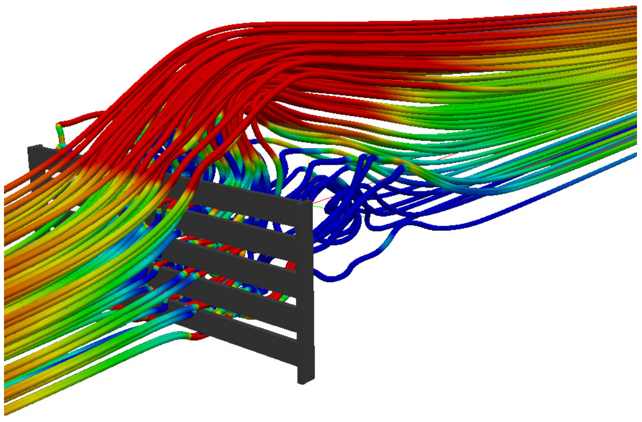
PRACE SoHPCContact
Leon, Kos
E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied
SIESTA, Quantum Espresso, SLURM, OpenSUSE, VMD, XcrysDen, VESTA, VirtualGL, VNCViewer, OpenMPI, OpenMP, Deneb, OpenBLAS, Atlas, Texmaker

PRACE SoHPCMore Information
SoHPC blogs
SoHPC

PRACE SoHPCAcknowledgement
I would like to thank the SIESTA[1] and QE[2] developers for their work as well as all other developers of any free open source software used.

PRACE SoHPCProject ID
1518



Turbulence with Open-FOAM

Benjamin Chapman

OpenFOAM is open source computational fluid dynamics (CFD) software primarily used for investigating turbulence phenomena. In this project the simulation of external dynamic cases for a wind barrier was undertaken and the results have been compared to previous results using the commercial CFD software ANSYS as well as experimental data.

Turbulence is an exciting research area for which there is no concrete mathematical model and analytical solution. To this end, computational techniques must be employed. Depending on the software used, it can take days to run a turbulence simulation on a single processor. Fortunately, due to the advent of supercomputers, complex simulations can be run with relative ease by using multiple processors.

So what is turbulence? Loosely speaking, it is a chaotic flow of a fluid characterised chaotic changes. It is responsible for many everyday phenomena, the most prevalent example is the shaking one sometimes experiences whilst in an aircraft which caused by the flow of air over the wings. Another example is plasma turbulence, one of the primary obstacles to fusion energy. This can enhance the transport of heat and particles out of plasma and inhibit fusion energy production.

The aim of this project was to test current turbulence models found in OpenFOAM for high Reynolds number flows. The Unsteady Reynolds Navier-Stokes (URANS) approach was used for modelling the turbulent separated flows. The equation which provides the building blocks for this model is the Navier-Stokes equation[1]

$$\frac{\partial \underline{v}}{\partial t} + (\underline{v} \cdot \nabla)\underline{v} = -\frac{1}{\rho}\nabla P + \frac{\partial \sigma_{ij}}{\partial x_j} + \underline{g}. \qquad (1)$$

Method

Firstly, it was necessary to run a test case in order to familiarise myself with the software. The motorbike test case which is provided with the OpenFOAM installation was chosen as the geometry is similar to that of the case which was to be run for my project. The case was successfully simulated using the simpleFoam solver (the first term of equation 1 is ignored). I was then given the necessary mesh files (basically a picture of the situation which the computer uses) to run a simulation of wind flowing through a tunnel over a barrier placed at an angle of 90 degrees to the horizontal. Certain files within the motorbike test case were changed to make them compatible with the barrier geometry, then the situation was run in simpleFoam. After successful completion the simulation parameters were modified in order to calculate the pressure, viscous, and porous forces across the barrier and also to time average the simulation variables. The simulation was then run using the pimpleFoam solver. This is a transient solver (first term in equation 1 is included) and as such, allows for much more accurate calculations of the forces. The process outlined above was repeated for two different barrier configurations in which the angles the barrier made with the horizontal were 60 and 45 degrees.

As well as the actual simulations, a small python code was written in order to parse the force file produced by OpenFOAM and test for convergence, and a small Gnuplot script was written to plot the forces based on the output of the python file.

Results

The drag and lift forces for each barrier configuration were successfully calculated and plotted for comparison with experimental data and data from ANSYS. BC1, BC2, and BC3 refer to barriers making angles of 90, 60, and 45 degrees to the horizontal respectively. This plot is shown in figure 1.

It can be seen that the lift forces as predicted by OpenFOAM are in good agreement with the experimental data, more so than the corresponding ANSYS results. The drag force for BC1 calculate using OpenFOAM is $\sim 3.5\%$ higher than the experimental result. Similarly, the same force for BC2 is $\sim 5\%$ lower than the experimental result. Both results are reasonably good, however the same cannot be said of the BC3 configuration. This is $\sim 15\%$ higher than the experimental value. This may be explained by the relative quality of the meshes used for each configuration; as the BC3 mesh was of much lower quality than the BC1 and BC2 meshes and it is widely accepted that OpenFOAM is significantly more susceptible to poor mesh quality than ANSYS.

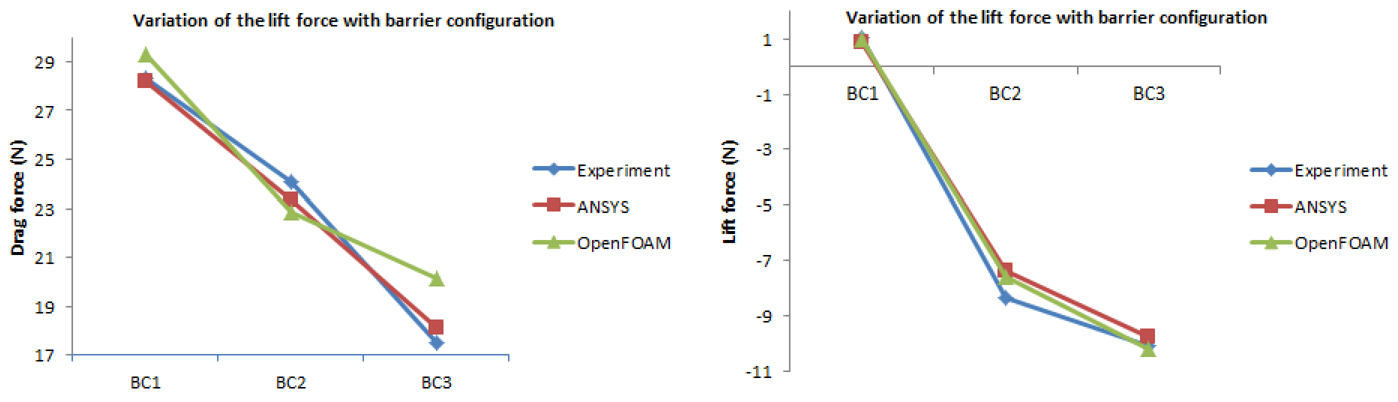


Figure 1: Lift and Drag forces for different barrier configurations

Using the 3D visualisation software ParaView, a slice of the 3D geometry in the direction of the wind flow was taken. This slice was used to produce 2 images, one of wind flowing over the barrier near the beginning of the simulation ($t = 200s$), and one at the end of the simulation ($t = 2000$). This are shown in figure 2.

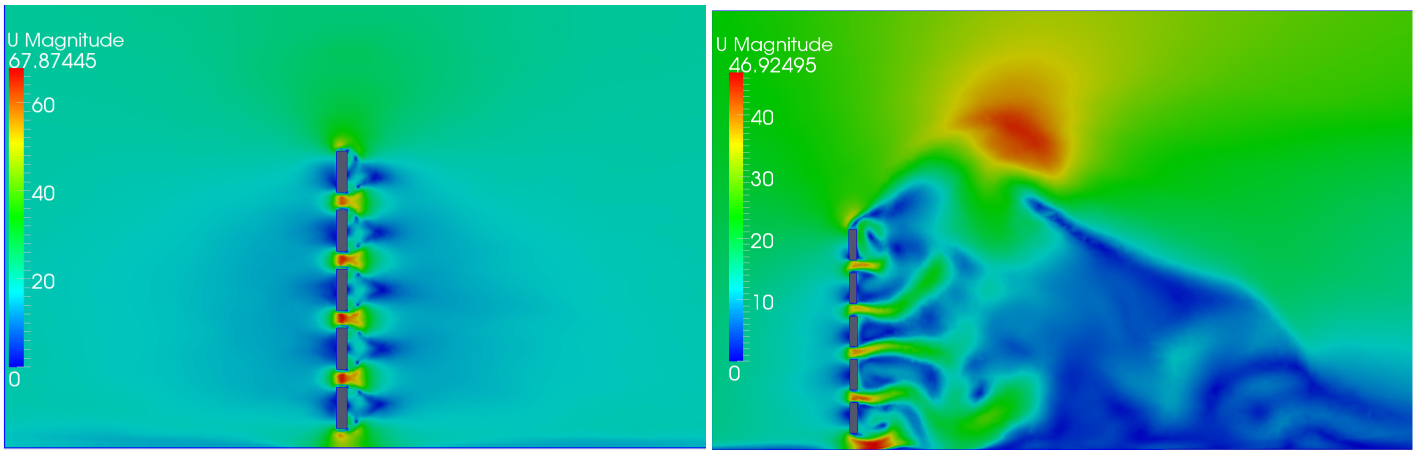


Figure 2: Velocity profile of the wind in the tunnel. The left images shows the flow at $t = 200s$ while the right image shows the flow at $t = 2000$.

It can be seen that the velocity of the wind slows down just before the barrier and is at its lowest value at the points just in front and behind the barrier. The wind velocity then speeds up again once it has passed the barrier.

Discussion & Conclusion

The results for the forces, particularly the lift forces are encouraging. ANSYS is expensive commercial software whilst OpenFOAM is free. If one can obtain scientific results of a similar quality using both, the latter seems the obvious choice for research institutions. The drag force results are less satisfactory, especially that of BC3. As previously stated, this is most likely due to poor mesh quality. If one were to do a similar project in the future, the first place to start would be to improve the quality of all 3 meshes (the BC1 and BC2 weren't perfect). Maybe using meshes with less errors in OpenFOAM will give results in better agreement with experiment and possibly improve on those obtained via ANSYS.

References

[1] G. J. Pert, Introductory Fluid Mechanics for Physicists and Mathematicians, Department of Physics, The University of York UK, Wiley, 2013.

PRACE SoHPCProject Title
Using open source CFD (OpenFOAM) for turbulence modelling of different wall-bounded flow regimes.

PRACE SoHPCSite
University of Ljubljana, Faculty of Mechanical Engineering, Slovenia

PRACE SoHPCAuthors
Benjamin Chapman, E-mail: bc661@alumni.york.ac.uk
[The University of Warwick,] UK

PRACE SoHPCMentor
Mario Telenta, University of Ljubljana, Faculty of Mechanical Engineering, Slovenia

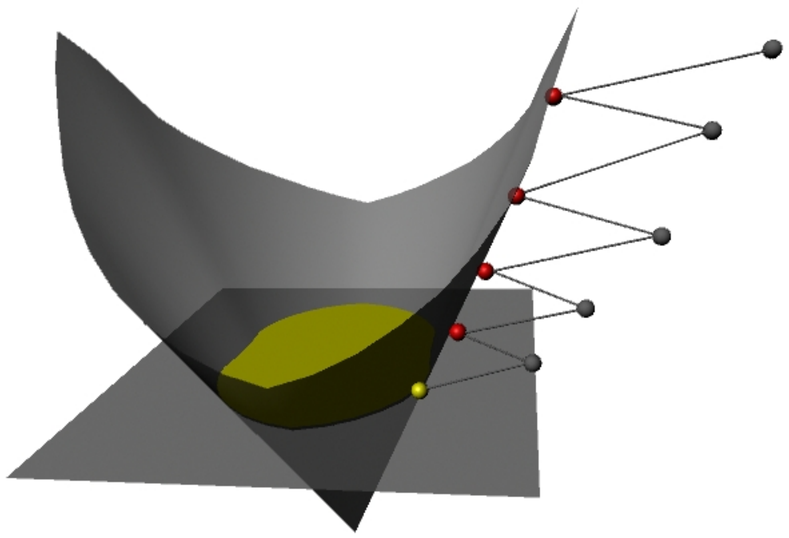
Benjamin Chapman

PRACE SoHPCSoftware applied
OpenFOAM, ParaView, Python, Gnuplot, Fortran90

PRACE SoHPCAcknowledgements
The author would like to acknowledge Dr. Mario Telenta and Dr. Leon Kos for there invaluable guidance.

PRACE SoHPCProject ID
1519

A parallel implementation of the boundary point method, an algorithm to solve semi-definite optimization programs



Parallel boundary point method

Mircea Simionica

The boundary point method has turned out to be an efficient algorithm for semi-definite programs with a large number of constraints. The project consisted of importing a Matlab code into a C++ parallel version for SPD problems characterised by a block diagonal structure. The final product will allow problems to be solved which are beyond reach for state-of-the-art methods.

Semidefinite programming (SDP) has experienced a big development in mathematical optimisation since the 1990's. It's main applications is combinatorial mathematics and control theory. Semi-definite optimisation is concerned with the study of a symmetric matrix that yields optimal value of a linear objective function and that satisfies a given number of linear equations. In mathematical terms a primal semi-definite program (PSDP) can be expressed as follows:

$$\begin{aligned} \max \quad & \langle C, X\rangle \\ \text{such that } & A(X) = b \\ & X \succeq 0 \end{aligned}$$

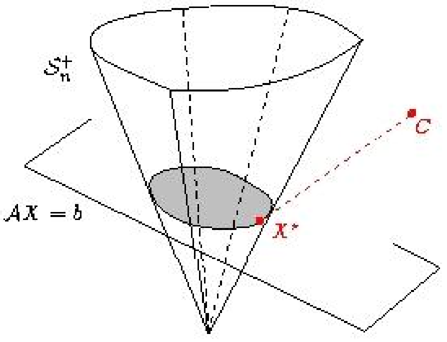


Figure 1: Cone of positive semi-definite matrices

The boundary point method is a relatively new algorithm that sets itself as an alternative to interior point methods. The algorithm operates on the dual formulation of the problem (DSDP):

$$\begin{aligned} \min \quad & b^T y \\ \text{such that } & A^T(y) - C = Z \\ & Z \succeq 0 \end{aligned}$$

Lagrange multipliers are applied to the dual equations, yielding the following Augmented Lagrangian:

$$L_\sigma = b^T y + \langle X, Z + C - A^T(y)\rangle + \frac{\sigma}{2}\left\|Z + C - A^T(y)\right\|^2$$

The steps that define the complexity of the algorithm are solving a large system of linear equations and computing a spectral decomposition of a symmetric matrix in each iteration. The pseudo-code of the algorithm is illustrated in *Figure 1*. The goal of the project was to deliver a C++ parallel version of an existing MATLAB code. The code is specifically prepared for semi-definite programs characterised by a block diagonal structure. i.e. involving matrices that can be split into smaller sub-matrices along the diagonal. The matrix below is an example of a block diagonal matrix.

$$A = \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ 0 & A_{22} & 0 & 0 \\ 0 & 0 & A_{33} & 0 \\ 0 & 0 & 0 & A_{44} \end{bmatrix}$$

This type of structure leads to a parallelisation based on the number of diagonal blocks. The parallel aspect focuses on the inner iteration: once the sparse linear system has been solved, the remaining steps are repeated according to the number of blocks. They are also independent so they can be solved concurrently. The serial code loops through the blocks, whereas the parallel version solves each block independently, speeding up the computations. This will allow to solve larger instances of semi-definite problems, defined by more diagonal blocks and with a bigger number of constraints, leading to new challenges for other methods.

The code is written in commented C++, MPI is used as message-passing system between processes and a manual is included, servind as documentation of the code. External linear algebra libraries are heavily used. Armadillo

```
Select   $\sigma > 0, \{\varepsilon_k\} \to 0, \varepsilon > 0$.
$k = 0; X^k = 0; Z^k = 0;$
repeat until   $\delta_{outer} \leq \varepsilon$    (Outer iteration for $k = 0, 1, ...$)
    repeat until   $\delta_{inner} \leq \sigma\varepsilon_k$    (Inner iteration: $(X^k, \sigma)$ held constant)
        Solve for $y^k$:   $AA^T(y^k) = A(Z^k + C + \frac{1}{\sigma}X^k) - \frac{1}{\sigma}b;$
        for each diagonal block $i$
            $W_i^k = A^T(y^k) - C_i - \frac{1}{\sigma}X_i^k;$
            $Z_i^k = W_+; V^k = -\sigma W_-;$
        end for
        $\delta_{inner} = \|A(V^k) - b\|;$
    end repeat
    $X^(k+1) = V^k;$
    $k \leftarrow k + 1;$
    $\delta_{outer} := \|Z^k - A^T(y^k) + C\|;$
end outer repeat
```

Figure 2: Parallel boundary point method pseudo-code

is utilised throughout the whole code, while the linear system is solved with Eigen's sparse solver. MATLAB (or Octave, freely available under the GNU license) is used for generating the input data. We spent the first week translating the code from MATLAB to C++. In the process Armadillo library turned out to be really helpful. Armadillo is an open-source, high quality linear algebra library useful for algorithm development. Its syntax is similar to MATLAB, so we saved time not implementing specific MATLAB features.

A few challenges

Some time was spent on a parallel implementation of the eigen decomposition using ScaLAPACK. This was useful since it allowed us to better understand how the block-cyclic distribution of data among the processes works. Me and my mentor soon realised that the sparse linear system was actually slowing the code down, so we decided to change route. We encountered most of the difficulties working on this stage of the project, since we had issues linking SuperLU, a direct sparse solver, to Armadillo.

First improvements

We experienced some first improvements once SuperLU was correctly linked, but the code was still under performing with respect to MATLAB. We knew that Eigen was another good linear algebra library so we decided to use one of its sparse linear solvers. Specifically, we went for Eigen's *SparseCholesky* module. This class provides Cholesky factorisations, which allow for solving $AX = b$ linear systems. In particular, the matrix A is factored as $A = LL^T$, where L is lower triangular. The system $LL^Tx = b$ is then solved with forward and back substitutions. This is good in terms of performance since the factorisation is computed only once, outside of the main loop and not in each iteration. Choosing to include Eigen objects in our code allowed us to deal with multiple linear algebra libraries, pushing us to search and find out methods of importing/exporting the data between different classes in an efficient manner. In fact, Eigen objects can be mapped to those of Armadillo. This avoids copying a huge amount of data in every iteration.

Parallel implementation

Solving the sparse linear system using Eigen rather than Armadillo considerably pushed the solve time down. The C++ serial version was now aligned to MATLAB's performance. We decided it was the right moment to begin the parallelisation phase. We focused on the for loop inside the inner iteration (*Figure 1*). Instead of computing several spectral decompositions in sequential, the calculations are now split among the processes and the current solution is updated locally.

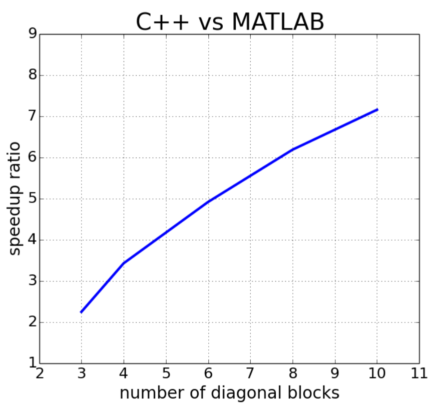


Figure 3: Speed-up ratios

Number of blocks	Matrix dimension	Number of constraints	Number of spectral decompositions	Degree of convergence	Time (seconds) MATLAB	C++
6	500	378456	32100	0.02558	3980.93	840.70
8	200	82824	42800	0.01662	641.20	106.40
8	400	325592	42800	0.02141	2924.00	505.20
10	100	26080	53500	0.02938	183.14	25.76
10	300	228410	53500	0.01648	2155.16	291.20

Table 1: Time comparison between MATLAB and C++ parallelized version

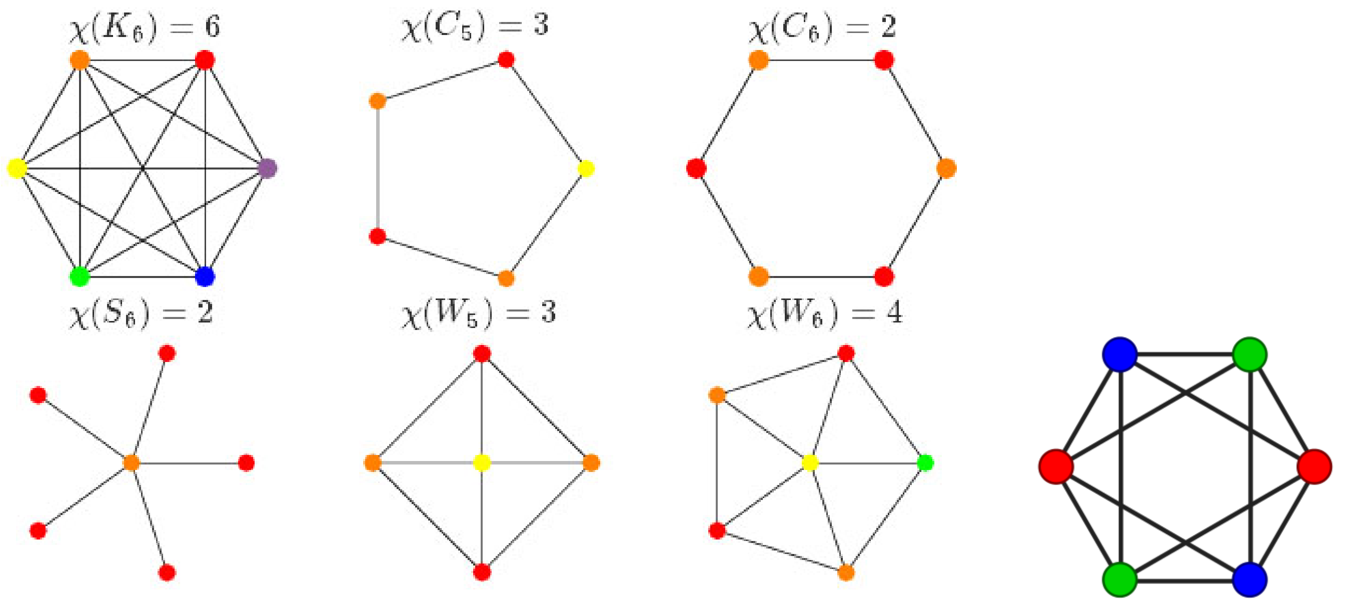


Figure 4: Examples of vertex colouring problems

Even more improvements

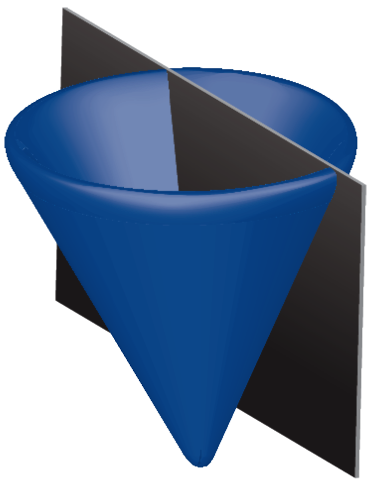
The parallel version looked promising. We performed some initial tests, alternating between increasing the number of blocks and the size of the matrix of coefficients. There was evidence of scalability with the number of blocks (hence with the number of processes). Naturally the code did not scale linearly because of communication overheads. The results were pointing in the right direction, but there was one more thing to do. Armadillo integrates with BLAS and LAPACK libraries. These are packages providing matrix operations and numerical linear algebra routines. Up to now Armadillo was linked against traditional BLAS and LAPACK packages. The biggest improvement in the code came the moment we linked Armadillo against OpenBLAS. OpenBLAS is a multi-threaded, high performance replacement of BLAS. This allowed the C++ serial code to compete with MATLAB's performance and made possible bigger speedup ratios running the parallel version. A sample of results can be observed in *Figure 3* and *Table 1*.

Applications

semi-definite optimisation programs equipped with a block diagonal structure are useful to investigate some problems in combinatorial mathematics. For example, in graph theory the chromatic number of a graph G is the smallest number of colours needed to colour the vertices of G such that no two adjacent vertices share the same colour. The chromatic number of a graph is usually denoted $\chi(G)$ and its computation is an NP-hard problem. Usually this quantity is bounded by other numbers obtained by solving SDP problems. One of them is $\Psi(G)$, whose computation requires polynomial time.

Future research

semi-definite programming is in general a strong tool to approximately solve very hard problems from optimisation: mixed integer linear and quadratic programming problems, a wide range of non-linear programming problems and also to detect positivity of commutative and non-commutative polynomials, an inspiring problem from real algebraic geometry. The parallel version will allow to tackle SDP instances that are beyond reach for the state-of-the-art methods for this type of problems.



References

[1] J.Povh, F. Rendl, A. Wiegele. A boundary point method to solve semi-definite programs. *Computing*, 78(3):277-286, November 2006

[2] J.Povh, J. Govorcin, N. Gvozdenovic. New heuristics for the vertex coloring problem based on semi-definite programming. *Central European Journal of Operations Researh*, 21(1):13-25, June 2013

[3] C. Sanderson. Armadillo: an open source C++ linear algebra library for fast prototyping and computationally intensive experiments. *Technical Report*, NICTA, 2010.

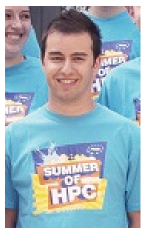
[4] Gaël Guennebaud and Benoît Jacob and others. *Eigen v3*. Retrieved from *http://eigen.tuxfamily.org*, 2010.

PRACE SoHPCProject Title
Parallel boundary point method

PRACE SoHPCSite
Faculty of information studies, Novo Mesto University of Ljubljana

PRACE SoHPCAuthors
Mircea Simionica, Bocconi University, Italy

PRACE SoHPCMentor
Janez Povh, Faculty of information studies, Novo Mesto, Slovenia



Mircea Simionica

PRACE SoHPCContact
Leon Kos, University of Ljubljana
E-mail: leon.kos@lecad.fs.uni-lj.si

PRACE SoHPCSoftware applied
C++, MPI, Armadillo, Eigen, OpenBLAS, MATLAB

PRACE SoHPCMore Information
HPC Prelog

PRACE SoHPCAcknowledgement
I would like to thank PRACE &. SoHPC for this summer experience. I acknowledge my mentor Janez Povh & the SoHPC coordinator Leon Kos for giving me the opportunity to learn and discover.

PRACE SoHPCProject ID
1520

www.summerofhpc.prace-ri.eu