EUROfusion

EUROFUSION WPISA-REP(16) 16104

R Hatzky et al.

HLST Core Team Report 2010

REPORT

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

HLST Core Team Report 2010

Contents

1. Executive Summary

1.1. Progress made by each core team member on allocated projects

In agreement with the HLST coordinator the individual core team members have been/are working on the projects listed in Table 1.

Project acronym	Core team member	Status
ASCOT-10	Nitya Hariharan	finished
EUTERPE	Nicolay Hammer	prolonged
GYNVIZ	Matthieu Haefele	running
ITM-EU4IA	Nicolay Hammer	finished
MGEDGE	Kab Seok Kang	finished
KINSOLD	Kab Seok Kang	prolonged
ZOFLIN	Nitya Hariharan	scheduled

Table 1 Projects mapped to the HLST core team members.

Roman Hatzky has contributed in particular to the projects ASCOT-10 and EUTERPE. Furthermore, he was occupied in management and dissemination tasks, e.g. the development of the HLST web site, due to his position as core team leader.

Matthieu Haefele worked on the GYNVIZ project. The aim of the GYNVIZ project is to unify and to provide support for the whole hardware and software chain concerned with the visualization process of large datasets being produced by the major European gyrokinetic codes.
Three main components can be identified. The first one consists of a uniform data format, namely XDMF. It has been decided to use this format as standard because a wide spectrum of data types can be expressed and its design is very flexible implying rather low effort on the code developer side. A collaboration with the XDMF team in the U.S. has been set up in order to improve parts of the implementation to our concerns. A new release is expected soon.
The second component consists of a suite of post-processing software. The main part is intended to turn 3D time-varying datasets in 4D compressed ones in order to explore them with 4D visualization. The 4D compression and visualization are technologies transferred from a former EUFORIA project. The post-processing tool is being developed within the GYNVIZ project and is nearly finished.
Finally, network and computing infrastructures hosted by the computer centers RZG and JSC are the third component. In close collaboration with RZG, a DEISA project involving JSC has been established and DEISA accounts have been created for each GYNVIZ user. As a result, data generated by HPC-FF can be easily and efficiently transferred from JSC to RZG via the DEISA file system with a simple “cp” command. Subsequently, a remote visualization session can be started on the newly built visualization cluster at RZG to explore the transferred data. This infrastructure is already up and running.

Nicolay Hammer worked primarily on the ITM-EU4IA and EUTERPE projects. In addition, he did some supplementary work to the former OPTGS2 project.

The aim of the ITM-EU4IA project was to port the Unified Access Layer (UAL) to HPC-FF. The UAL is a software library developed by the Integrated Tokamak Modelling (ITM) task force of EFDA. It is designed to serve as a unified communication platform for numerical physics codes which are used in Europe to simulate the different physical aspects of tokamak fusion devices. A working local installation of the UAL library package was generated and tested on HPC-FF. However, an internal component of the UAL the so-called MDSplus library does not

work properly. Accordingly, the problem has been passed to the MDSplus developers from ITM.

The EUTERPE code is a global gyrokinetic particle-in-cell code aimed at simulating turbulence in fully 3D stellarator geometry. In a first step the code was successfully ported to HPC-FF. Different problems like e.g. a memory leak in the Parastation MPI had to be fixed. Next we targeted the performance improvement of the EUTERPE code by making use of the Single Instruction, Multiple Data (SIMD) capabilities of the Intel Nehalem processor also known under the term of vectorization. Such SIMD capabilities will be further extended in future CPU design e.g. by Intel AVX (Advanced Vector Extensions) and are in line with the Single Instruction, Multiple Thread (SIMT) concept of Graphics Processing Units (GPUs). The two main particle loops of EUTERPE had to be restructured for vectorization. In addition, it was necessary to get detailed information of the performance of the vector registers. Corresponding results were achieved with a self programmed test bed of common intrinsic functions both on Intel's Nehalem and IBM's POWER6. Detailed results of these tests have been published in the HLST report "*Performance Tuning Using Vectorization"* as a contribution for training of young scientists to the use of upcoming new computer architectures. Unfortunately, in the special case of the EUTERPE code, the performance improvement due to vectorization was cancelled by its own overhead so that the overall performance did not change significantly. However, the new code structure made a more detailed performance analysis possible which revealed further potential for performance improvement. Currently we are focusing on optimizing the access time to the look-up tables of the magnetic field data.

Nitya Hariharan worked primarily on the ASCOT-10 project to reduce the memory consumption of the code. The code uses a large amount of memory to store histogram data for diagnostical purpose, i.e. data which have been produced by the binning of Monte Carlo particles. We have used linked lists to implement a new data compression algorithm for the histogram data using a sparse format which significantly reduces memory consumption and also allows the user to use up to seven dimensional histograms. The input magnetic field data, represented by splines, also consume a large amount of memory per MPI process. We have used Shared Memory Segments (SMS) to represent the spline data. The user can now keep a single copy of the magnetic field data within each node on HPC-FF and share it among the MPI processes within that node. With the new much more efficient memory management the ASCOT code is able now to simulate more detailed magnetic field configurations which results in more accurate physical simulations.

Some further work was done on the JOREK-HR project. We were able to find out that the problems we faced during porting of JOREK to HPC-FF were due to the level of thread support available on ParTec MPI. ParTec has now provided to our request a working version of the library and JOREK has been benchmarked successfully on HPC-FF by HLST member Florent Sourbier.

The benchmark of the former BEUPACK project was revisited and we investigated the effect of having more memory and bandwidth, per core, on the benchmark codes. Codes that use a large amount of memory seem to benefit to some extent.

Kab Seok Kang worked on the MGEDGE and KINSOLD projects.

In the MGEDGE project, we have worked on the efficient implementation of the multigrid method in the GEMZ (Gyrofluid ElectroMagnetic) code of Bruce D. Scott which solves nonlinear gyrofluid equations for electrons and one or more ion species in tokamak geometry. Starting from an already existing implementation of the multigrid method we amended the intergrid transfer operators and the linear solver. We further continued with detailed adaptation and testing of the implemented multigrid algorithm on the VIP machine at RZG and the HPC-FF machine at JSC.

Over all, we proved that our implementation of the multigrid method using the conjugated gradient method as a "lowest level" solver and first-order intergrid transfer operators has very good strong and weak scaling properties up to 2048 cores. Thus, it is suitable for usage on massively parallel machines like HPC-FF. For details please see the HLST report *"Parallelization of the Multigrid Method on High Performance Computers"* which has been published as IPP report 5/123.

The Particle-in-Cell (PIC) code BIT1 is restricted so far to 1D3V plasma and 2D3V neutral particle modeling with a reasonable scaling up to 1000 and more processors. Hence, ongoing work is focused on enhancement of the code to 2D3V plasma simulations of the Scrape-Off-Layer (SOL). However, the Poisson solver in 2D has been identified as a bottleneck for the scaling properties. So the work plan is to develop a good scaling Poisson solver in 2D either with the multigrid method itself as a solver or as a multigrid preconditioner for a PGMRES solver. The method as it has been tested so far has still some limitations. Good convergence of the multigrid and PGMRES solvers is achieved so far only for an inner empty space and internal conducting area which matches the coarsest grid. Hence, it will be of interest how these limitations can be overcome.

1.2. Further tasks and activities of the core team

1.2.1. Dissemination

Haefele, M.: Post-processing and visualization: general issues and some solutions, *Theory Meeting*, 2nd – 5th November 2010, Ringberg, Germany.

Hatzky, R.: The High Level Support Team, *Munich Computational Science Center Meeting (MCSC): Kooperation Anwenderbetreuung*, 3rd February 2010, Garching, Germany.

Hatzky, R.: The High Level Support Team, *IFERC Special Working Group 1 (SWG1),* 16th February 2010, Garching, Germany.

Hatzky, R. and Günter, S.: The EFDA HPC project, *Inertial Fusion Energy "Keep-in-Touch" (ITE-KiT) Meeting,* 22nd March 2010, Madrid, Spain.

Hatzky, R. and Günter, S.: The EFDA HPC project, *ITM-TF meeting,* 15th September 2010, Lisbon, Portugal.

Hatzky, R. and Günter, S.: The EFDA HPC project, *GoTiT Training Course on Modern Programming and Visualization Techniques*, 18th – 29th October 2010, Garching, Germany.

1.2.2. Training

Haefele, M.: CEMRACS scientific event - Numerical Modelling of fusion, Research session, 26th – 30th July 2010, Marseille, France.

Haefele, M.: Parallel I/O, *RZG seminar*, 4th October 2010, Garching, Germany.

Haefele, M.: GoTiT Training Course on Modern Programming and Visualization Techniques, 18th – 29th October 2010, Garching, Germany.

Haefele, M.: Comparison of Different Methods for Performing Parallel I/O, HLST web site, [online], 2010.

Hammer, N.J. and Hatzky, R.: Combining Runge-Kutta discontinuous Galerkin methods with various limiting methods, IPP report 5/124, Max Planck Society eDoc Server, [online], 2010.

Hammer, N.J.: Combining Runge-Kutta discontinuous Galerkin methods with various limiting methods, GOTiT e-Seminar, April 14th 2010, Garching, Germany.

Hammer, N.J.: Performance Tuning Using Vectorization, IPP report 5/126, Max Planck Society eDoc Server, [online], 2011.

Hariharan, N.: HPC-FF - Overview and Experience, *GoTiT Training Course on Modern Programming and Visualization Techniques*, 18th – 29th October 2010, Garching, Germany.

Hariharan, N.: HPC-FF - Overview and Experience, *GOTiT e-Seminar*, December 15th 2010, Garching, Germany.

Hatzky, R. and Bottino, A.: Particle-in-Cell methods in plasma physics, *HLST seminar,* 11th November 2010, Garching, Germany.

Kang, K.S.: Parallelization of the Multigrid Method on High Performance Computers, IPP report 5/123, Max Planck Society eDoc Server, [online], 2010.

1.2.3. Workshops and conferences

Arter, W., Barnes, M.A., Roach, C.M., Knight, P., Hammer, N.J., and Hatzky, R.: Optimisation of the GS2 Gyro-kinetic code, *2010 International Sherwood Fusion Theory Conference*, 19th – 21st April, Seattle, USA: 2010.

Haefele, M., Navaro, P., Kos, L., and Sonnendrucker, E.: Euforia Integrated Visualization, 18th Euromicro International Conference on *Parallel, Distributed and Network-based Processing (PDP 2010),* 17th – 19th February 2010, Pisa, Italy.

Haefele, M.: La problématique du post-traitement, introduction à HDF5 et XDMF, *Workshop "Masse de données : I/O, format de fichier, visualisation et archivage"*, 13th January 2011, Lyon, France.

Hatzky, R.: HPC Simulations of Microturbulence in Fusion Plasmas, *International Supercomputing Conference' 10,* 2nd June 2010, Hamburg, Germany.

Hatzky, R. and Bottino, A.: Particle-in-Cell methods in plasma physics, *European-US Summer School on HPC Challenges in Computational Sciences,* 4th – 7th October 2010, Acireale, Italy.

Kang, K.S.: On finite volume multigrid method, *European Multi-Grid Conference EMG 2010,*19th – 23th September 2010, Isola d'Ischia, Italy.

1.2.4. Internal training

The HLST core team has attended:

- The HLST meeting at IPP, 14th January 2010, Garching, Germany.
- PRACE Workshop on "New Languages and Future Technology Prototypes" at LRZ , 1st – 2nd March 2010, Garching, Germany.
- 5th VI-HPS Tuning Workshop at TUM, 7th – 9th March 2010, Garching, Germany.
- Colloquium in the framework of a Symposium on "New Trends in Numerical Methods for Plasma Physics" at IPP, 8th July 2010, Garching, Germany.

- The HLST meeting at IPP, 29th September 2010, Garching, Germany.

Roman Hatzky and Kab Seok Kang have attended:

- *Theory Meeting*, 2nd – 5th November 2010, Ringberg, Germany.

Matthieu Haefele has attended:

- IPP Summer University on Plasma Physics and Fusion research, 20th – 24th September 2010, Garching, Germany.

1.2.5. Visits to collaborators

Nitya Hariharan has visited the ASCOT group:
Aalto University of Science and Technology, April 25th – 7th May 2010, Helsinki, Finland.

1.3. Recommendations for the year 2011

The call for the use of High Level Support Team resources will close on 31st January, 2011. The core team leader will present a recommendation on how to distribute the work load over the HLST members. The corresponding spreadsheet will be passed to the HLST coordinator. The final decision will be made by the HPC board.
Until the new projects are approved by the HPC board Nicolay Hammer and Kab Seok Kang will, in agreement with the HLST coordinator, continue with the EUTERPE project and the KINSOLD project, respectively. Nitya Hariharan will start at 1st February with the ZOFLIN project.

2. Report on HLST project GYNVIZ

The aim of the project is to unify and to provide support for the whole hardware and software chain that comes from the current codes' output to the interactive and remote visualization software. Fig. 1 presents a global picture over the whole project.

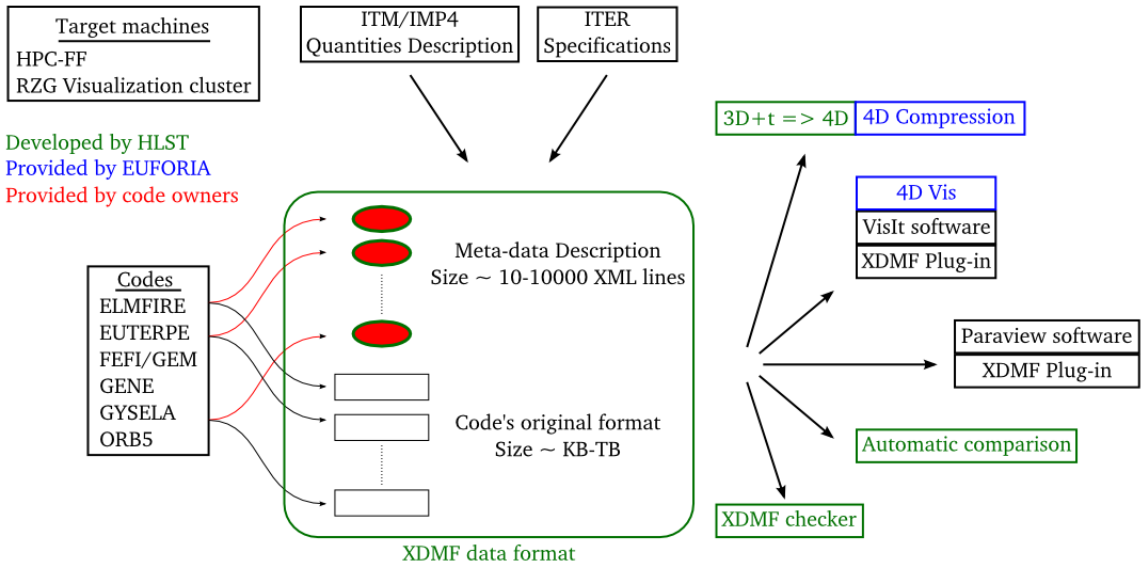

Fig. 1 Overview of the overall project

2.1. File format technology review

From the visualization point of view, the existing software VisIt, Paraview, Ensight, Avizio and AVS Express are the main ones in this field. They are now mature technologies and cover almost all visualization requirements for the GYNVIZ project. So the key issue is the definition of a common data file format. It has to be technologically up to date and it must be understood by visualization software (at least VisIt and Paraview).

The latest versions of several libraries have been tested. They can be split into two categories. The first one focuses on computer science objects and is mainly composed of HDF5 and NetCDF4. Both are C libraries that can be called mainly from Fortran, C and C++. They are intended to create portable binary files containing regular arrays of data. The second category focuses on computational modeling objects and provides for the user a higher level of abstraction interface. It aims at describing, with meta-data, how the actual data represent computational meshes or variables that are defined on some meshes. When this kind of library actually writes this data/meta-data it can make use of either HDF5 or NetCDF4.

Table 2 shows the different libraries that have been tested and the criterion that have been used to evaluate them. In general, the marks represent how a technology fits with the purpose of the project and go from 1 (very bad) to 5 (very good). Details of the criterion and general comments on each technology can be found below.

Library (category)	Writing technology	Relia.	Durabi.	Port.	Intrus.	Vis	Test	Eff.	Mark
HDF5 (1)	Kernel, STD lib	5	5	5	2-4	2	5	2	4
NetCDF4 (1-2)	STD lib, HDF5	5	4-5	?	2	1	1	2	2
Silo (2)	HDF5	5	5	4	1	3	5	4	3
Exodus II (2)	NetCDF4	?	?	?	1	5	1	5	1
XDMF (2)	STD lib, HDF5	4	3	5	4	4	4	2	4

Table 2 Evaluation of the different technologies.

Reliability: 5=very good, 1=very poor
Durability, long term maintenance: 5=very good, 1=very poor
Portability on different systems: 5=very good, 1=very poor
Code intrusive: if such a technology should be embedded into existing codes, how intensive will be this intrusion. 5=very small, 1=very large
Visualization software support: is the technology already supported by several visualization programs. 5=fully supported by many programs, 1=not supported
Level of Test: how far we have evaluated the technology. 5=very far, 1=only documentation reading
Efforts required in order to adapt/extend this technology for the project purpose: 5=no effort, technology can be used as it is, 1=big effort.
The Mark is simply a rating of the different technologies according to our context: 5=very good, 1=bad**.**

HDF5 is becoming a standard for writing data in different computational sciences. However it is not a format that is able to describe high level concept, only basic datasets are supported in the different visualization software.
NetCDF4 was a competitor of HDF5 and HDF5 seems to have won even if NetCDF provides some more high level representations.
Silo is a very good high level library. However, data must go through the silo interface in order to create a silo file and it can be quite intrusive from the code point of view.
Exodus2 is a well established format in the finite element community and it is very well supported. However, it really focuses on unstructured meshes which are not the general case in our context.
XDMF separates clearly meta-data from data and then introduces a very nice flexibility at different levels. Although, it is supported by a wide range of visualization software, the implementation is not perfect and the roadmap of this format is unclear.

XDMF technology offers several advantages. The first one is that the migration from the existing format of the different codes to the unified one will be easier with XDMF than with HDF5. With HDF5, either codes' I/O are left untouched and a probably costly post-processing step becomes mandatory, or, codes' I/O need to implement the HDF5 format. With XDMF, the different codes will only need to write on disk a small text file in addition to the current data file. This text file describes, with XDMF XML syntax, how the data are structured within the data file. As a result, the current format of the data file in the different codes is kept intact and the need for post-processing step and the development of a post-processing tool has become obsolete. Another advantage is that the plug-ins for different visualization software do not need to be developed as XDMF plug-ins already exist for several visualization software tools.

On the other hand, the technology review has shown some reliability and durability issues. That is why we have started a collaboration with the XDMF team in the USA in order to improve some aspects of their technology. The idea is to implement a semantic checker tool that checks the consistency between the XML description of the data and what is actually in the data file. On the end-user side, it will help to identify and to correct any semantic errors. On the developer side, this tool will help to validate/correct all the XDMF examples and accordingly will help to correct all the existing XDMF plug-ins. HLST and XDMF teams agreed on the facts that HLST will develop the checker tool, the XDMF team will correct the plug-ins and the examples validation/documentation will be done together. So, instead of investing the man power in the development of post-processing tools and the plug-ins, it has been shifted to this collaboration.

At the time of writing this report, the collaboration with the XDMF team starts smoothly. A first version of the XDMF checker has been implemented and has successfully been tested by the XDMF team. The XDMF team is currently developing a new interface for accessing data in XDMF format from different programming languages. A pre-alpha version has been tested and looks promising.

2.2. Visualization cluster at RZG

The visualization cluster at RZG has entered its production phase middle of October 2010. Reservation system, collaborative work within a single session and use of 3D visualization software are the main functionalities that have already been successfully tested[1]. Installation of the EUFORIA 4D visualization plug-in for VisIt is ongoing work.

2.3. Data transfer between JSC and RZG computer centers

As the codes run on HPC-FF, they produce data on HPC-FF. In order to be visualized on the RZG visualization cluster, data have to be transferred from JSC at Jülich to RZG, Garching. The dedicated 10 Gb/s DEISA network is used for that purpose. As the DEISA filesystem is now mounted on the GPFS nodes in Jülich, it is possible, with an appropriate DEISA account, to transfer data from the HPC-FF Lustre filesystem to Garching GPFS filesystem. As the DEISA filesystem is also mounted on the visualization cluster, users can access directly their data. We are still in the process of getting every GYNVIZ members equipped with such a DEISA account.

2.4. GYNVIZ tool development

2.4.1. Automatic comparison tool

It aims at comparing either the results from the same code for two different runs or the result of two different codes that simulate the same physics test case. Three main issues have to be handled. Firstly, relevant physical quantities have to be selected and an agreement on their definition has to be found among the different codes. This issue is not addressed within GYNVIZ but some work has been done within the Integrated Modelling Project 4 (IMP4) of the Integrated Tokamak Modeling (ITM) initiative. Secondly, data have to be written in the same format and this is exactly the purpose of the GYNVIZ project. Finally, comparison methods have to be applied to decide if the results agree or not. As a first implementation, only norms of the difference of the 1D quantities are computed. This can be extended in the next phase

[1] http://www.rzg.mpg.de/computing/hardware/miscellaneous/hp_vizcluster

of the project. A first version of this tool is already implemented and is waiting for testing.

2.4.2. Visualization

Fig. 2 shows a screenshot of the 4D visualization tool displaying a 4D particles distribution function f(r,theta,phi,vpar). It shows four 2D slices of the function: from top to bottom and left to right, we have r-theta slice, r-phi slice, r-vpar slice and phi-vpar slice. To define completely the r-theta slice for example, one has to give a value to phi and vpar coordinates. These values are shown on the phi-vpar slice accordingly on the position of the white cross. The whole idea of this 4D visualization method is to enable the user to move interactively these crosses in the different 2D views in order to explore interactively the 4D volume. The challenge is to refresh interactively these slices. This is achieved by 4D compression and parallel processing.

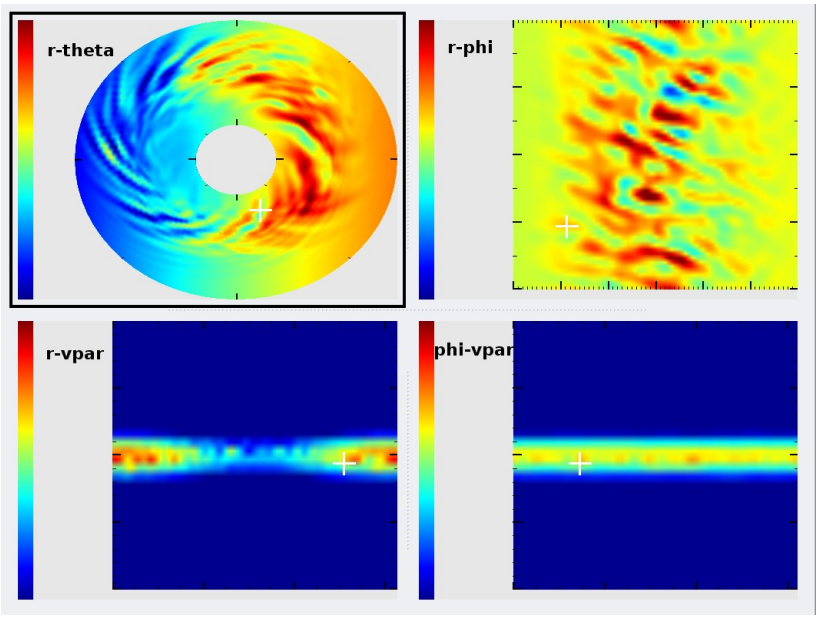

Fig. 2 Screenshot of the 4D visualization method

This visualization method has been developed within the EUFORIA project as a VIsIt plug-in. It is planned to install this plug-in on the RZG visualization cluster.

2.4.3. 3D + t → 4D conversion tool

The first implementation of this tool will focus on the conversion of 3D time varying data in XDMF format into 4D compressed data. This will enable the possibility to interactively explore the dataset both in space and time using several 2D slices. This tool will be built on top of libraries and software developed by the EUFORIA project. The development is nearly finished.

2.5. Parallel I/O study

The current trend of computational power growth is mainly based on the increase in the number of cores. This phenomenon modifies deeply the way applications should be programmed. Indeed, to benefit from this computational power, applications must be parallelized and must scale very well. In particular, applications' I/O will definitely become an issue on the next generation of machines. Although this topic is not the core of the GYNVIZ project, it is strongly related as the data must be first written on disk before their visualization.

The purpose of the study is to evaluate 13 different I/O methods that write a 2D array on disk from a distributed application using a block-block distribution layout. Evaluations have been performed on two different architectures: HPC-FF at JSC (Lustre file system) and VIP at RZG (GPFS file system). Interpretation of the results and programming advices are given in a separate report available on HLST website[2]. This report has been disseminated within HLST, the GYNVIZ project, RZG and EUFORIA.
The study has improved our knowledge on parallel I/O in general and in particular on HPC-FF. This is of key interest for next generation computers, especially IFERC.

2.6. Conclusions and future work

The visualization cluster is up and running, the data transfer mechanism is now settled and each GYNVIZ member is in the process of getting an appropriate DEISA account to access both the cluster and the data transfer mechanism. Most of the tools planned to be developed are already implemented or nearly finished. The only missing piece is the new version of the XDMF format. The pre-alpha version being tested so far seems to be promising.
After having a first working version of the whole framework, the next step will consist in teaching GYNVIZ users how to bring their data into the XDMF framework. It is planned to give individual support to each group and to guide them until they are autonomous. Depending on the feedback achieved by helping the GYNVIZ users, the continuation of the project will follow one or several of the following ways: to extend post-processing tools based on XDMF, to continue the improvement of XDMF format by implementing a test suite and to go into more detail in the parallel I/O study.

[2] http://www.efda-hlst.eu/training/HLST_scripts/comparison-of-different-methods-for-performing-parallel-i-o/at_download/file

3. Supplementary Report on HLST project OPTGS2

3.1. Introduction

The major long-term objective of the GS2 [1] code developers is to resolve the performance bottleneck of the gyrokinetics code, which arises from the implicit finite difference scheme used in the solver. As discontinuous Galerkin finite element methods (DG-FEM) became quite popular in the last decade, it was suggested by the project coordinator, Wayne Arter, that this method could resolve the aforementioned numerical obstacle.

The basic aims and results of the first parts of project OPTGS2 can be found in the HLST annual report 2009 which covers the topics: basic properties and performance of DG-FEM schemes, control of numerical artefacts using various limiter methods, and solving the (1+1)D GS2 model problem using a DG-FEM scheme.

Another long-term objective of the code developers, which is expected to be on a much longer time scale than the primary objectives, is to integrate an adaptive mesh refinement method into the GS2 code. Since this is a large project involving many development steps, the aim of this part of the project was simply to test some of DG-FEM's prospects concerning grid complexity and adaptivity.

Since the GS2 code solves a time dependent problem, suitable time integration schemes for the DG-FEM spatial discretisation method are of great interest in the OPTGS2 project as well. Here the aim of the project is to investigate the behaviour of the DG-FEM in combination with simple Eulerian time discretisation as well as with explicit and implicit higher order time discretisation methods. Thus, the strengths and weaknesses of the time discretisation schemes regarding efficiency and accuracy should be explored.

For further details please see the HLST report "*Combining Runge-Kutta discontinuous Galerkin methods with various limiting methods*" [2].

3.2. Implicit Runge-Kutta time integrators

Higher order explicit time integration schemes require a rather restrictive time step ($\Delta t \sim 0.1$) for well resolved numerical results [3]. Thus, it was of interest to investigate higher order implicit time integrators for the DG-FEM method, especially, higher order implicit Runge-Kutta (IRK) time discretisation schemes. We implemented implicit Runge-Kutta schemes of different types: Gauss-Legendre IRK schemes up to four stages and eighth order accuracy, multistage IRK schemes of Radau type up to fifth order and finally two fourth order accurate singly-diagonally-implicit Runge-Kutta (SDIRK) schemes.

In our first approach in the DG-FEM-IMPLICIT solver the IRK equation system was solved using a fixed point iteration method, which has some disadvantages. In contrast to the Eulerian backward scheme the implemented IRK schemes are not unconditionally stable when solved iteratively. Moreover, with increasing number of stages of the IRK scheme and increasing size of the time step, the number of iteration steps for the fixed point iteration increases drastically, especially for the fully implicit IRK schemes, i.e. the Gauss-Legendre and the Radau IRK scheme. On the contrary, those schemes are unconditionally stable for very large time steps, when the solution of the IRK equation system is obtained directly by matrix inversion. This is possible for linear differential operators as in the case of the linear advection equation.

To include the IRK scheme into matrix inversions for implicit schemes by the used direct solver package (LAPACK), the matrix vector form of the implicit solver had to be rewritten in a block-matrix-vector form. Afterwards, both the directly and the iteratively solved IRK schemes were successfully tested with our standard advection equation test problem used already for the explicit solver.

However, all implicit schemes, independent whether solved directly or iteratively, showed with increasing time step growing spurious oscillations. Although, these wiggles can be controlled to a certain extent with the moment limiter methods [4], this fact sets an upper limit to the size of the time step which can be practically used. Although, some higher order IRK schemes were able to produce equivalent accurate results as higher order explicit schemes, in any case the implicit methods were numerically more expensive. This statement persists, even though solving the equation system of the fully implicit schemes directly is more efficient than solving them iteratively. Especially due to the practical time step limit mentioned above the ability to use larger time steps for IRK methods compared to ERK methods is not feasible.

3.3. Non-equidistant moving grid

The first step of this part of the project was to study the prospect of the DG-FEM to handle non-equidistant grids. Therefore, a grid with globally modulated grid cell size, as well as a locally refined grid were implemented in the DG-FEM solver and tested successfully. Here “globally modulated” denotes that the size of the grid cells was defined by a global sine function having a periodicity of the grid length. In contrast to that “locally refined” means here that single grid cells were split into smaller grid cells which fit in total into the size of the replaced grid cells. The implemented DG-FEM solver [5] produced good results when solving the one dimensional linear advection equation on non-equidistant grids.

For any modification of a numerical grid at run-time, an additional routine is needed which can project it onto the polynomial basis of the modified grid. We have implemented such a routine and tested it successfully.

Afterwards, both methods described above were tested together. However, the results delivered by this non-equidistant moving grid method are not very promising. The remapping scheme using the polynomial bases on both, the old and the new grid, is equivalent to a polynomial interpolation scheme with the same order as the used polynomial bases. The interpolation process introduces additional numerical viscosity due to the fact that the interpolated solution is smeared out over neighbouring grid cells by the interpolation process.

The method was finally tested with a setup using a grid which was refined at the position where steep gradients were present in the initial conditions and which was co-moving with the advection velocity. The setup for the comparison calculations was using a static grid having approximately the same resolution as the co-moving grid in the non-refined parts. In the end, the numerical solution of the advection equation using the co-moving grid with refinement and the solution using a static non-refined grid did not differ significantly. The reason seems to be again the introduced numerical viscosity by the remapping scheme. Since the new HLST projects were allocated in March 2010, the work was stopped at this point with the acceptance of the project coordinator W. Arter.

3.4. References and applicable documents

[1] GS2 homepage at sourceforge.net
http://gs2.sourceforge.net

[2] HLST Report: Combining Runge-Kutta discontinuous Galerkin methods with various limiting methods
http://edoc.mpg.de/display.epl?mode=doc&id=446381

[3] HLST annual report 2009, Sec. 4.4.1

[4] HLST annual report 2009, Sec. 4.4.2

[5] HLST annual report 2009

4. Final Report on HLST project ITM-EU4IA

4.1. Introduction

The aim of this project was to port the Unified Access Layer (UAL, [1]) to HPC-FF, which was installed in the mid of 2009 at the Jülich Supercomputing Centre (JSC).

The UAL is a software library developed by the Integrated Tokamak Modelling (ITM) task force of EFDA. It is designed to serve as a unified communication platform for numerical physics codes which are used in Europe to simulate the different physical aspects of tokamak fusion devices. HPC-FF will become the testing and production platform for fusion simulations in Europe, therefore, it is essential to make the UAL available on HPC-FF.

This report will give the final status of this project. Additional information, experiences, and remarks on the topic can be found in the more detailed final report of project ITMEU4IA [2].

4.2. Final status of the library porting

The following parts of the current UAL version, i.e. version 4.08a, have been successfully ported to HPC-FF:

- Low Level Library based on the MDSplus Library
- C++ interface
- FORTRAN interface
- Python interface
- Java interface

To do so, additionally the following software packages had to be installed on HPC-FF, either locally or globally. The packages are listed below:

- MDSplus 2.2 (www.mdsplus.org)
- Java Development Kit (JDK6) 1.6.022 (http://jdk6.dev.java.net)
- Blitz++ 0.9 — a C++ Scientific Library (www.oonumerics.org/blitz/)
- HDF5 (recompiled) (www.hdfgroup.org/HDF5/)
- SZIP compression library (www.compressconsult.com/szip/)
- g95 FORTRAN Compiler (www.g95.org)
- SWIG — The Simplified Wrapper and Interface Generator (www.swig.org)

All listed UAL library parts have been compiled and tested successfully with a local installation.

The UAL requires a locking mechanism on the file system level. Such a locking mechanism, in case of a Lustre file system this is FLOCKING, is not provided on HPC-FF, because it causes I/O performance penalties. Moreover, the Lustre users' guide disadvises the usage of FLOCKING explicitly, thus it will be most likely not provided at all.

However, there is a mechanism called LOCAL FLOCKING which could be used instead. It acts as a kind of pseudo locking mechanism having no or only a slightly negative impact on the performance of the Lustre file system. For that purpose on our request a testbed with a Lustre file system partition supporting the LOCAL FLOCKING mechanism had been set up on a test cluster system at JSC. It differs from HPC-FF just in size, i.e. the total number of nodes. Using this setup the UAL passed the I/O tests which come with the UAL package without any problem. As a

result, LOCAL FLOCKING has now been enabled on the Lustre file systems useable on the JUROPA/HPC-FF cluster.

4.3. Conclusions and additional comments

The proposal estimated that the porting of the UAL library package could be done within the project time span of three months. However, it turned out that many problems and issues had to be solved for that aim. Ultimately, the HLST project provided the developers of the UAL with detailed feedback on the UAL's portability to other environments.

Finally, a working local installation of the UAL library package was generated and tested. A test shot file was generated and filled using test software written in C and FORTRAN. However, due to internal errors of MDSplus it was not possible to read the newly generated shot files, even if, the same shot files could be read after transferring them to the ITM gateway cluster. The problem will be further investigated by the MDSplus developers.

A final project meeting was held together with the UAL development team, where an oral presentation on the results was given. The UAL source code modified during this project was checked-in in a new branch of the UAL SVN repository and a TAR archive of the local installation of MDSplus and the UAL was given to the project collaboration.

4.4. References and applicable documents

[1] G. Manduchi, F. Iannone, F. Imbeaux, G. Huysmans, J. Lister, B. Guillerminet, P. Strand, L.-G. Eriksson, and M. Romanelli, *Fusion Engineering and Design* **83**, 462 (2008), Proceedings of the 6th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research.

[2] HLST Report: HLST Project ITMEU4IA - Porting the ITM UAL to HPC-FF
http://www.efda-hlst.eu/internal/reports/annual-hlst-report-2010/reports-of-the-projects-2010/itmeu4ia-report/view

5. Report on HLST project EUTERPE

5.1. Introduction

The EUTERPE code is a global gyrokinetic particle-in-cell code aimed at simulating turbulence in fully 3D stellarator geometry. It was created at the Centre de Recherches en Physique des Plasmas (CRPP) [1] and afterwards developed further at the Max-Planck-Institut für Plasmaphysik (IPP) [2]. Beside these institutions it is currently used as well at the Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT) in Madrid.

The performance improvement of the EUTERPE code started with a scan for potential improvements of the single processor performance. For identification of the most CPU time consuming routines the code has been instrumented by the simple but efficient *perf* library. The *perf* library was programmed by the RZG scientist Reinhard Tisma and gives information about the time spent and the Mflop rate achieved in a detected region. Therewith, two most work intensive parts have been identified. It is the part which prepares the particle pushing (FORTRAN subroutine *PUSH*) and the part which does the charge assignment (FORTRAN subroutine *CCASSIGN*) for the different orders of B-splines., e.g. for quadratic B-splines (subroutine *QUAD_ASS*).

These subroutines, including calls to further subroutines inside, have been chosen to be targeted for optimization by Single Instruction, Multiple Data (SIMD) capabilities of the Intel Nehalem processor. The Nehalem processor supports a number of SIMD instructions by the SSE4.2 instruction set to access small-scale SIMD with 128 bit registers. The Streaming SIMD Extensions (SSE) is a SIMD instruction set extension to the x86 architecture, designed by Intel and introduced in 1999 in their Pentium III series processors. The capability of SIMD in Intel and AMD processors will be further extended in the future as Intel has already announced the Advanced Vector Extensions (AVX) which will be the new 256 bit instruction set successor to SSE and is designed for applications that are floating point intensive. In addition, modern Graphics Processing Units (GPUs) are using Single Instruction, Multiple Thread (SIMT) implementations, capable of branches, loads, and stores on 128 or 256 bits at a time. If a code has been programmed for SIMD it will automatically run on a SIMT architecture but not the other way round. Hence, it is of key interest to investigate today's Nehalem's SIMD capabilities for realistic double precision (64 bit) problems to be in line with future hardware development.

5.2. Cleaning, porting and testing the code

The code is written in Fortran 95 and consists of approximately 30,000 code lines localized in about 20 different files. Compilation is done with the GMAKE utility which makes it necessary to define the dependencies between the different source files. Hence, it is very important that the dependency list is updated during code development. To prevent overlooking some dependencies the so-called Automake utility from Polyhedron has been used. It analyzes the set of source files and constructs automatically a consistent dependency list. Minor issues in the original dependency list have been corrected.

In addition, the static Fortran source code analyzer FORCHECK was used to find errors and development debris which had been piled up over code development. Many improvements and corrections have been suggested leading to a better readability and portability of the code.

After testing the code with the direct solver from IBM WSMP on HPC-FF it became obvious that cases with larger grids could only be run by using the option

dist_solver=1. In this mode the matrix is distributed over all clones and thus the code can use the whole distributed memory to store the Cholesky factorization. However, a segmentation fault occurred in the ordering routine of WSMP. After constructing a separate test case the problem was reported to the lead developer of WSMP, Anshul Gupta and finally solved.

Consistency checks of the MPI usage have been done by the MARMOT tool. No errors could be detected. However, it became clear that the usage of the MPI standard was not consistent. The code has a pre-processor switch to select between exclusive usage of the MPI routines of the MPI-1.2 standard and the MPI-2.1 standard, respectively. Unfortunately, this selection turned out not to be rigorous. It has been corrected and the MPI-2.1 standard usage has been made the default.

The code crashed unexpectedly during long simulations on HPC-FF. After applying a memory diagnostic it became obvious the code suffered from a memory leak. Surprisingly, the leak was not caused by the code itself but instead by the Parastation MPI from ParTec. Tests using Intel MPI clearly showed that there was no memory problem. After consultations with the lead developer of the Parastation MPI, Jens Hauke, the problem could be identified. The bug was corrected in the Parastation MPI version psmpi2-5.0.22-1. Later it was found that the ORB5 code on HPC-FF had also suffered from the same problem which had been circumvented by using Intel MPI instead of Parastation MPI.

Test runs with an executable compiled with the *-check all* option of the Intel Fortran compiler *ifort* had revealed two uninitialized variables in the *BCPART* routine which now have been initialized correctly.

5.3. Reduction of communication overhead

In the EUTERPE code, several global sums are performed as MPI_ALL_REDUCE operations. These global sums are very often executed consecutively. Thus, small chunks of data are communicated over the whole network several times. Instead, it is much more efficient to assemble the data first in a buffer which is then communicated as a whole. By doing so, a few tens of global sums could be saved. In addition, it is not necessary to provide an MPI_ALL_REDUCE operation if an MPI_REDUCE is already sufficient. This is the case if data are only summed up for diagnostic purpose to be printed out or to be written on file by the master process. Accordingly changes in the code have been made.

5.4. Source code changes to enhance the vectorizability

In the case of the original subroutine *QUAD_ASS*, already some parts of the source code were vectorized by the auto-compiler unit. However, the impact onto the total execution time of the *QUAD_ASS* subroutine is negligible. In the subroutine *PUSH,* the auto-vectorizer does not vectorize the source code at all. This is due to the fact that these subroutines are dominated by loops over all particles residing on one core. The content of the loops is very long and complicated with many calls to other subroutines and makes it unsuited for vectorization. Thus, the original subroutines had to be restructured for vectorization.

In a first step the main particle loop inside the *VEC_QUAD_ASS* and *VEC_PUSH* subroutines, which are clones of the subroutines *QUAD_ASS* and *PUSH*, respectively, has been split into an outer loop providing chunks of vector data and an inner loop working on those chunks. The size of those data chunks can be controlled by a parameter. Thus, the outer loop enables a complete control over the amount of work load which will be processed by the SIMD capabilities, i.e. vector registers, of the CPU.

In a subsequent number of working steps, the inner loop was split into several smaller block loops. Accordingly scalar variables used inside the former main loop have been exchanged by vector variables. This is mandatory for vectorization, no matter whether done by the auto-vectorizer of the compiler, e.g. the auto-vectorizer of the Intel Fortran compiler *ifort* [3] or done manually by the programmer, using e.g. the Vector Mathematical Functions (VMF) of the Intel Math Kernel Library (MKL) [4]. This is due to the fact that on the one hand, statements which shall be replaced by a vector version have to be isolated within the source code. On the other hand, the auto-vectorizer unit of the compiler has certain limitations when analysing complex code structures. Even if the code structure has the potential for vectorization it could simply fail because the source code structure is simply too complex for the auto-vectorizer.
In general, making use of the auto-vectorizer of a compiler is the preferred way in the first place, since no compiler dependent code structures are required. However, there exist code structures where the auto-vectorizer fails or may not achieve the expected performance (compare [5]). In those cases it is preferable to manually implement the vectorization concepts and to pay the price of generating compiler dependent code which restricts the universality of the code [6]. Furthermore, in some cases it is advantageous to use compiler vectorization pragma statements. They can help the compiler to analyse source code statements and can correct for misinterpretations concerning the efficiency of the vectorization [3].

5.5. Performance study of vectorization on the Intel Nehalem versus the IBM POWER6 architecture

Generally, vectorization can only be advantageous if a significant speed-up is achieved in the vectorized loops. This is necessary to compensate the overhead generated by introducing vector variables which have to be read in and written out for each vector operation statement block. Hence, the memory access of vector processors is highly optimized for such operations to minimize such overhead. However, this is not the case for RISC processors like the Intel Nehalem and IBM POWER6.

Before changing the subroutines *QUAD_ASS* and *PUSH* for vectorization it was necessary to get detailed information of the performance data of the vector registers of Intel's Nehalem and IBM's POWER6 hardware. Especially the speed-up factor between the scalar and vector registers is of interest. Hence, test cases for common intrinsic functions as e.g. SQRT, SIN, LOG, etc., as well as some other statements and operations of interest, e.g. packing operations, have been investigated in great detail. Of basic interest is the dependence of the speed-up factor as a function of the vector length of double precision (64 bit) input data.

In the presented case of a vector multiplication we achieved the following result on the Intel Nehalem CPU shown in Fig. 3. Here the measurements without vectorization (Intel novec) are compared with cases were the vector registers have been involved. For this purpose we used the auto-vectorizer of the Intel Fortran compiler suite [3] (Intel autovec) together with the vector functions of the Intel MKL [4] library (Intel MKL VMF) and the IPP (Integrated Performance Primitives) [7] library (Intel IPP-Vec). In general it shows that best results on the Intel Nehalem can be achieved by the auto-vectorizer. Corresponding tests on the IBM POWER6 architecture have been done with the IBM XL Fortran compiler [8] together with the IBM Math Acceleration Sub-System (MASS) [9] version optimised for the IBM POWER6 architecture by linking with *–lmassvp6*.

In the following we concentrated on the most efficient results of the vector registers. For numerically relatively cheap intrinsic functions this is reached typically for vector

lengths approximately between 50 and 1000 when the data still fit into the private L1 cache of each core. For such cases the ratios of execution time on the scalar units in comparison to the vector registers have been calculated for different intrinsic function statements on the Intel Xeon X5570 and the IBM POWER6 CPU. The complete study is summarized in Table 3.

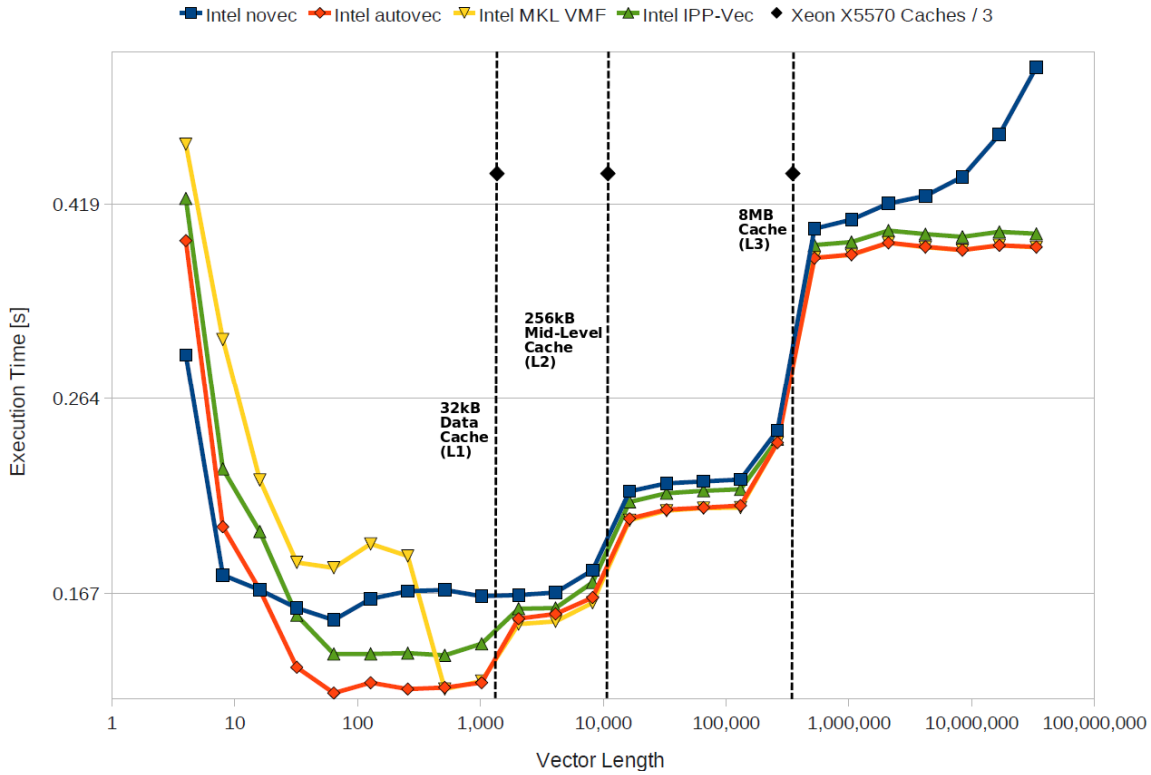

Fig. 3 Run time of a vector multiplication versus vector length compiled with the *ifort* compiler shown for a non-vectorized loop (blue) and loops vectorized by auto-vectorizer (red) and manual implementation of the Vector Math Functions in the Intel MKL (yellow) and Intel IPP (green) library on the Intel Nehalem (Xeon X5570) CPU of HPC-FF. The basic structure shows inefficient behaviour for very small loop sizes, followed by a broad minimum in execution time for vector sizes fitting into the Data Cache (L1). For larger vector sizes the behaviour is dominated by the cache hierarchy which leads to a step like structure. To emphasize this, the effective cache sizes of the Intel Xeon X5570 per used vector have been plotted in terms of double precision (64 bit) vector arguments as black dashed lines. For a multiplication, two arguments and one result vector are used and therefore the effective cache size is one third of the actual cache size.

It shows that on the Intel Nehalem the gain using the vector registers for the intrinsic functions under consideration can be between 0.75 and 2.51 with an average value of 1.6. This is remarkable because for some intrinsic function there is actually a loss of one quarter when using the vector registers. In contrast to this one always gains a speed-up on the IBM POWER6 which is between a factor of 1.01 and 6.37 with an average of 3.3 (we did not take into account the *ATAN2* result). So on average the speed-up on the IBM POWER6 is around a factor two higher than on the Intel Nehalem. The reason cannot be the length of the vector registers which is, on both architectures, 128 bits. Instead the programming of the vector registers on the IBM POWER6 seems to be compared to the standard mathematical library implementation more efficient as on the Intel Nehalem.

Further details please see the IPP report "*Performance Tuning Using Vectorization*" [5].

Function	HPC-FF (Intel Xeon X5570)	VIP (IBM POWER6)	HPC-FF/VIP scalar operations
Abs	1.26	—	—
Add	1.21	—	—
Atan2	1.49	(41.0)	0.09
Div	1.82	1.33	1.75
Exp	2.22	5.35	0.66
Floor	2.33	—	—
Int	1.00	—	—
Lim-Min	0.75	1.01	0.38
Madd	1.72	—	—
Min	1.00	1.01	0.57
Modulo	1.87	—	—
Mult	1.19	—	—
Nint	1.00	—	—
Pow	1.92	6.37	0.51
Pow3o2	2.03	—	—
Real	1.64	—	—
Sin	2.51	5.94	0.75
Sqr	1.40	—	—
Sqrt	2.23	2.15	1.76
Vpackm	1.00	—	—

Table 3 Listing of the ratios of execution time on the scalar units in comparison to the vector registers for different double precision intrinsic function statements on an Intel Nehalem, i.e. Xeon X5570 CPU (left column) and an IBM POWER6 (middle column) CPU, respectively. For this purpose, we used the scalar and vector execution times corresponding to the vector length which yields the overall smallest execution time. The right column shows the ratio between the execution time on the scalar unit on both, the Intel Nehalem and the IBM POWER6 CPU. Please note that not all intrinsic functions listed here were available on the vector pipeline of the IBM POWER6 and that the scalar computation of the Atan2 on the IBM POWER6 is relatively slow.

5.6. Performance results of vectorized routines

A speed-up of the vectorized subroutines *VEC_QUAD_ASS* and *VEC_PUSH* can be only expected if the loss due to overhead of the vectorization can be overcompensated by the gain by the vectorized parts of the routines. One part of the overhead is introduced by the loop splitting and the use of vector variables as described in Sec. 5.4 which causes many read and write operations on the L1 cache instead of reusing values in the registers of the CPU. The other part is caused by the data structure of the particles which was optimized for cache reuse and not vectorization. The first index loops over the attributes of each particle, as e.g. position in phase space x, y, z, v_par and mu, the second index loops over the particles. This data structure pays off essentially when the particles are communicated between cores. Instead, it is necessary for vectorization to have for each particle's attribute its own vector array. Hence, the data have to be reordered by a copying procedure which causes overhead.

To make sure that the auto-vectorizer recognizes the parts of the code to be vectorized the code has to be restructured in an appropriate way. This can be a delicate task where experience is required. However, if the code structure becomes too complex a vectorization can become impossible. For that reason, in both

subroutines, *VEC_QUAD_ASS* and *VEC_PUSH*, there exists still a significant fraction of source code which cannot be vectorized. These parts are especially look-up tables for the equilibrium quantities of the magnetic field which are stored in a large 4-dim array. Here the problem is the memory bandwidth as this array usually does not fit into the L1 and L2 cache levels. Other strategies than vectorization are necessary to improve the performance in these parts of the code.

After all these changes for vectorization have being made the overall performance could not be increased for a typical test run of 128 million ions and electrons on 64 cores of HPC-FF. Strictly speaking, the vectorized subroutine *VEC_QUAD_ASS* is running 4% slower than the non-vectorized *QUAD_ASS* subroutine and the vectorized subroutine *VEC_PUSH* is running 1% slower the than the non-vectorized subroutine *PUSH*. The gained speed-up by vectorization was not able to overcompensate the introduced overhead. The main reason seems to be the relatively small length of the vector registers on the Intel Nehalem CPU which can only store two double precision data simultaneously. Hence, the speed-up gained by vectorization is limited. Compared to the size of classical vector super computers, e.g. the NEC SX series with a vector (register) length in the order of 256 vector elements, the vector registers of both, the Intel Nehalem architecture and the IBM POWER6 architecture, having a total length to store two double precision numbers are small.

Also the results on the IBM POWER6 architecture are not encouraging. The vectorized subroutine *VEC_QUAD_ASS* is running 1% faster than the non-vectorized *QUAD_ASS* subroutine and the vectorized subroutine *VEC_PUSH* is running 1% slower than the non-vectorized subroutine *PUSH*. Here an additional problem is that the auto-vectorizer of the XLF IBM compiler seems to have problems to recognize all structures which have been recognized for vectorization by the Intel *ifort* compiler. Hence, it would be necessary to introduce direct calls to the MASS vector library by hand. We will avoid this strategy as this would produce code being exclusively optimized for the IBM environment while we have performance optimization for the HPC-FF computer in our focus.

However, the restructuring of the dominant subroutines VEC_QUAD_ASS and VEC_PUSH of the EUTERPE code carried out so far are an investment into the future. On the one hand, the size of the hardware vector registers will grow already within the next generations of hardware architectures. And on the other hand, the vectorization capabilities of the compilers will get more sophisticated. In addition, the contributed HLST report “Performance Tuning Using Vectorization” [5] gives detailed information on how different intrinsic functions react to the auto-vectorizer. Thus, the vectorization capability of the SIMD instructions of the Intel Nehalem CPU can achieve under other circumstances, than being present in the PIC code EUTERPE, a significant speed-up.

5.7. Future plans

Unfortunately the vectorization of the dominant routines in EUTERPE did not increase the performance of the code. However, the new structure of the vectorized particle loops offered the chance of a detailed performance analysis. It shows that especially the calls to the look-up tables for the equilibrium quantities of the magnetic field are very costly due to the memory access speed. This means that the dominant routines are memory bound. Hence, the data locality should be increased. One possibility would be to copy the relevant equilibrium quantities needed in a certain subroutine into a work array. Then it would be possible to gain all equilibrium quantities at the particle position with one call instead of calling the interpolation routine for each quantity separately. For better cache reuse this could be done for thousands of particles in one sweep. This would also decrease the subroutine calling

overhead introduced by separate calls for each particle. A speed-up of the routines in the range of more than ten per cent seems to be achievable.

5.8. References and applicable documents

[1] G. Jost, T.M. Tran, W.A. Cooper, L. Villard, K. Appert., Global linear gyrokinetic simulations in quasisymmetric configurations. Physics of Plasmas **8**: 3321, 2001.

[2] V. Kornilov, R. Kleiber, R. Hatzky, L. Villard, G. Jost, Gyrokinetic global three-dimensional simulations of linear ion-temperature-gradient modes in Wendelstein 7-X**.** Physics of Plasmas **11**: 3196, 2004.

[3] Intel® Fortran Compiler User and Reference Guides, Chapter: Automatic Vectorization Overview
http://software.intel.com/sites/products/documentation/hpc/compilerpro/en-us/fortran/lin/compiler_f/ index.htm

[4] Intel® Math Kernel Library for Linux* OS User's Guide, Chapter: Vector Mathematical Functions
http://software.intel.com/sites/products/documentation/hpc/compilerpro/en-us/fortran/lin/mkl/refman/ index.htm

[5] N. J. Hammer, HLST Report: Performance Tuning Using Vectorization
http://edoc.mpg.de/get.epl?fid=74364&did=536180&ver=0

[6] Aart J. C. Bik, The Software Vectorization Handbook, Chapter 9, Intel® Press 2004
http://www.intel.com/intelpress/excerpts/excerpt_vmmx3.pdf

[7] Intel® Integrated Performance Primitives for Intel® Architecture, Reference Manual, Volume 1: Signal Processing, Chapters: Essential Functions, Fixed-Accuracy Arithmetic Functions
http://software.intel.com/sites/products/documentation/hpc/ipp/index.htm

[8] Language Reference of XL Fortran for AIX, V13.1
http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp?noscript=1

[9] MASS vector libraries for AIX -- Overview
http://www-01.ibm.com/support/docview.wss?uid=swg27018490

6. Final report on the ASCOT-10 project

During the period 25th April to 7th May, Nitya Hariharan's visit to Helsinki was scheduled to get used to the ASCOT code and to familiarize with the procedure to create the necessary input files, execute the code and check results. Also, discussions on the project requirements were to be done to get an overall view of how the project should be approached in the best possible manner. The initial work plan was to refactor the code to make use of local variable paradigm instead of a global variable paradigm. This would provide the possibility to use shared memory parallelism like OpenMP. Using OpenMP would, in turn, enable ASCOT to simulate more detailed magnetic fields and also improve the runtime of the existing simulations.

Use of OpenMP not only improves efficiency, in certain cases, but also reduces memory consumption. However, it can also be intrusive in the sense that one needs to determine parts of code that can be parallelized using threads and data that can be shared among the threads. We found that the runtime benefits to the code with both MPI and OpenMP was not much and so decided to look at other parts of ASCOT to find out if the memory consumption could be reduced without much intervention to the code. We found that the program used large datasets of magnetic field data as input and during the post-processing phase made use of a large amount of memory to store the histogram data for diagnostical purposes. The histogram data are typically produced by binning the Monte Carlo particles being distributed over the phase space. Hence, we have concentrated our efforts to improve the memory consumption of the code for the input and output data. Incorporating OpenMP into ASCOT was thus no longer required.

While trying to port the code to HPC-FF, we encountered some FPEs (Floating Point Exceptions). Following our feedback to the ASCOT team about the FPEs, they looked into the code to fix the issues.

6.1. Histogram module

The original version of ASCOT stored the histogram data in multi-dimensional arrays. Given that a seven dimensional histogram consumes a large amount of memory, it made it imperative that we provide a solution that requires the minimal amount of memory.

The histogram module for the ASCOT code would provide substantial savings of memory and also allow data to be collected for up to seven dimensions. Some features of the module included creating histograms with user specified data such as min and max in each dimension, number of slots in each dimension etc., copy, free and summing up of the histogram data. In addition to that, the module would also have I/O routines that could be used to write out the histogram data into a file in a portable format.

The histogram module is also general enough that it can be used for other future projects as well. The code has been put into the SVN repository under the ASCOT-10 directory for future HLST use.

6.1.1. Linked list

Since, the histogram data that has to be binned has a sparse structure, using linked lists to store the histogram data is an optimum solution to save memory. We only need to store those index values at which the data has been binned. In addition, we need to store the weight at each index point.

A linked list consists of a set of nodes. Each node in the list, in turn, is a derived data type containing information. In our case, the derived data type consists of the index and weight values of the histogram. A pointer from one node in the list to the next allows for traversing the list. A 1D linked list is the simplest case of a linked list with a header node that points to the start of the list. Each node then points to the next node in the list. A 2D linked list has a matrix structure with nodes representing rows and columns. Fig. 4 and Fig. 5 give a simple example.

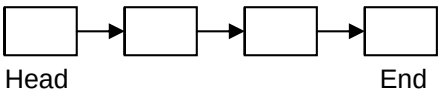

Fig. 4 A 1D linked list with four nodes.

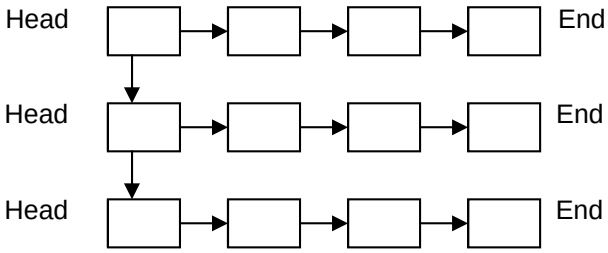

Fig. 5 A 2D linked list.

A linked list is a sparse representation of a one or multi-dimensional array. Implementing a 1D linked list is fairly simple. The level of complexity increases for a 2D linked list, and it is very expensive to implement and maintain higher dimensional linked lists. However, we can use certain properties of the index value to simplify the representation of a multi-dimensional array as a linked list.

An index into an array can be represented as a tuple. This is generated by knowing the index and the size of the array in each dimension. For example, in an array of size nxm the tuple for index (1, 2) will be 1 + (2-1)*n = 5. This tuple value is always unique for a set of index values. Since the size of each dimension of the histogram is known, we can, therefore, store the index into a multi-dimensional array as a tuple and use a 1D linked list structure to store the histogram data. This in effect means that we compress the multi-dimensional array into a 1D linked list which makes it easier to maintain. We also build the linked list in a sorted order so that it is easy to insert values into the list.

One of the disadvantages of using a sorted linked list is, that the time taken to traverse and insert a new node in a linked list increases linearly with the number of nodes. We need to compare the index value to be inserted, with each value in the linked list till we reach the correct position in the list. To get around this problem, we use a pointer to the node that was last inserted. We also ‘split’ the linked list into four quadrants, according to the number of nodes, by maintaining pointers that point to each quadrant of the linked list. Then, while inserting a new node, we compare its index value with the value of the last inserted node and with the values in the quadrant nodes. Hence, instead of searching the list from the start, we start from the best possible location and scan only the remaining parts of the list. We also update the positions of the quadrant pointers when the list has grown by a certain size, i.e., number of nodes.

We have implemented a 1D and a 2D linked list. Instead of using tuples for a 2D list or the structure in Fig. 5, we have implemented an improved version by creating an array that represents the number of rows in the histogram. Fig. 6 shows the structure of our 2D linked list implementation. The columns in each row are represented by the linked list. A pointer to the head of each linked list is stored in the array. This reduces the number of pointers to be maintained and also makes traversing the list easier. The 2D linked list is used only for the 2D case. Histograms with three and higher

dimensions have their index values stored as a tuple, and the structure of a 1D linked list is used. We also maintain the list in a sorted order of the index values.

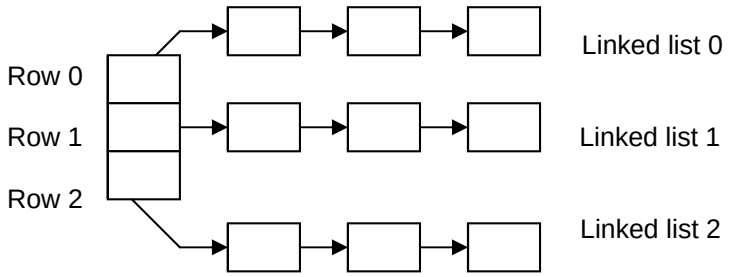

Fig. 6 An alternate implementation of a 2D linked list.

6.1.2. Merging the local linked lists

The histogram data needs to be written out in a file for post-processing. In order to write out the histogram data, we need to sum up the contents of the histogram on each processor, at a single processor. In other words, we need to merge the contents of the linked lists. This requires communicating the contents of a linked list between processors which is a non-trivial task. Since linked lists make use of derived data types with pointer components, the memory location is no longer contiguous like an array. This has two implications

1) Functions like MPI_Reduce can no longer be used due to the sparse structure of the data. The assembly of the histogram data over all processors is no longer straightforward.
2) Use of pointers to the nodes of the list means that the contents of the linked list need to be packed before sending it to another processor.

For the first case, we make use of a binary tree structure to reduce the data at one processor. To do so, we create MPI communicators in advance that define the processors involved in communication at each stage. For 2^n processors this is trivial. However, we have also provided functionality for cases where the number of processors is not a power of 2. In such cases, we define an extra communicator at a level where we have an odd number of processors. This includes the master processor and the highest ranked processor. Fig. 7 shows a simple example with seven processors. The levels indicate the stage of data reduction.

We create an extra communicator at Level 2, for processor 0 and 6. At Level 3, we create two communicators for processors 0, 2 and 0, 4.

Finally we loop over all these pre-defined communicators to reduce the data at one processor. To deal with the second case, we make use of the fact that MPI allows the user to create user-defined structures for a data type. Also MPI has the MPI_GET_ADDRESS function which returns the address of a variable. We can use this to calculate the displacement of each value stored in the linked list and create an MPI Struct. This can then be sent to the receiving processor in the form of an array. The sending processor determines the rank of the receiving processor using the communicator to which they belong at the level the communication is taking place.

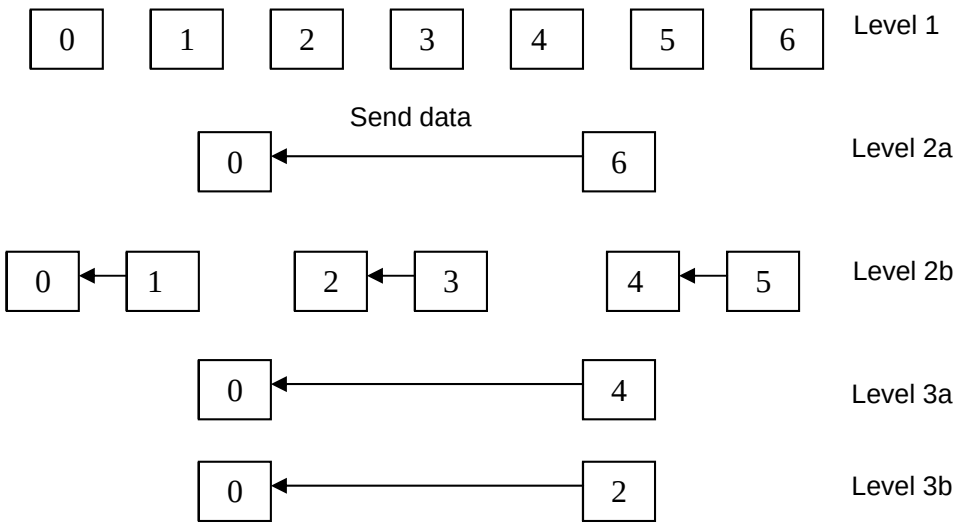

Fig. 7 MPI communication among processors.

At the receiving end, the array is merged back into the linked list contents of the receiving processor. If an index value is already present in the linked list, then the corresponding weight value from the array is added to the linked list node. A new index value is inserted into the list when the index value is not present. We can, at this point, make use of the pointers to the different locations in the linked list to speed up the merging process as well. This however, will be done only for the 1D linked list structure. For the 2D structure, we have the row index that we can use to get to the appropriate linked list and then scan the list with the column values to get the right position into the list.

In addition to the normal histogram, we have also implemented vector histograms. These are used to simultaneously store the data for up to ten histograms with the same bin size, min and max values etc. This has the advantage that we only need to calculate the index into the histogram once, and then use a vector of weight values to store each value into the histograms.

6.1.3. I/O

The output data i.e., the histogram data has to be written out in a format that is portable. The user should be able to load the histogram data in another program and analyze it. We have decided, in agreement with the ASCOT team to make use of the HDF5 file format. It provides a high level interface for Fortran 90 programs and is also readable by Octave, the JET alternative for MATLAB.

HDF5 files allow complex data types, for example, derived data types to be written out as datasets. ASCOT requires a simple dataset that includes the index and weight values. The meta-data of the histogram, for example, the name of the histogram, its dimensions, name of dimensions, units, min and max of each dimension, etc., are written out as attributes and attached to a dataset. HDF5 provides tools such as *h5dump, hdfview* that can be used to look at the contents of an HDF5 file. The user can write out multiple histogram data into a single HDF5 file.

The histogram module has been delivered to the ASCOT group and Simppa Äkäslompolo, who is the contact person for ASCOT, was able to carry out initial tests with it. They are in the process of restructuring their code to make it more modular and are adding more features to it. They plan to incorporate the histogram module into the new version of ASCOT, i.e. ASCOT4, once it is completed. We have, meanwhile, done some stress tests on the module and found it to be stable. The

code has also been analyzed using the FORCHECK analyzer to remove any bugs. Further performance tests will be done on the module by the ASCOT group.

6.2. Shared Memory Segments

The other major improvement that we were able to suggest to the ASCOT group, in terms of memory usage, was the use of Shared Memory Segments (SMS). These are areas of memory that can be shared by processes within the same SMP node. In the case of HPC-FF, this essentially means that up to 8 cores/processes within the node can share a particular area of memory.

SMS are a part of the System V IPC framework that provides techniques to communicate data among different processes running in one or more computers connected through a network. SMS have the advantage that they are easier to incorporate into an MPI program than OpenMP. Also, they do not require the use of a thread-safe MPI library. This justifies their use in the ASCOT code as well instead of using OpenMP.

FORTRAN and C interoperability is a part of the FORTRAN 2003 standard that allows a FORTRAN variable to be represented as a C pointer and vice versa. This allows the user to declare a variable in FORTRAN and convert it to an equivalent C variable. Since the IPC framework has subroutines in the C language, use of this feature of FORTRAN 2003 makes it easier to use SMS in FORTRAN.

Dr. Ian Bush, from NAG, has written a FORTRAN IPC module (FIPC) [1] that makes use of the System V IPC framework and the interoperability between FORTRAN and C. This creates a SMS segment from a FORTRAN 1D or multi-dimensional array and returns a pointer to it. The module makes use of the MPI library to create a context similar to the MPI_COMM_WORLD context in MPI. This context, called the *fipc_ctxt_world*, is a structure that contains different communicators including all processes (similar to MPI_COMM_WORLD), all processes in each node, and a designated 'Master' process in each node. The structure is shown diagrammatically in Fig. 8. Similar to the *mpi_init* and *mpi_finalize* calls, the module also has *fipc_init* and *fipc_finalize* calls to create and destroy the *fipc_ctxt_world* context. More information about the module can be found in his report [1].

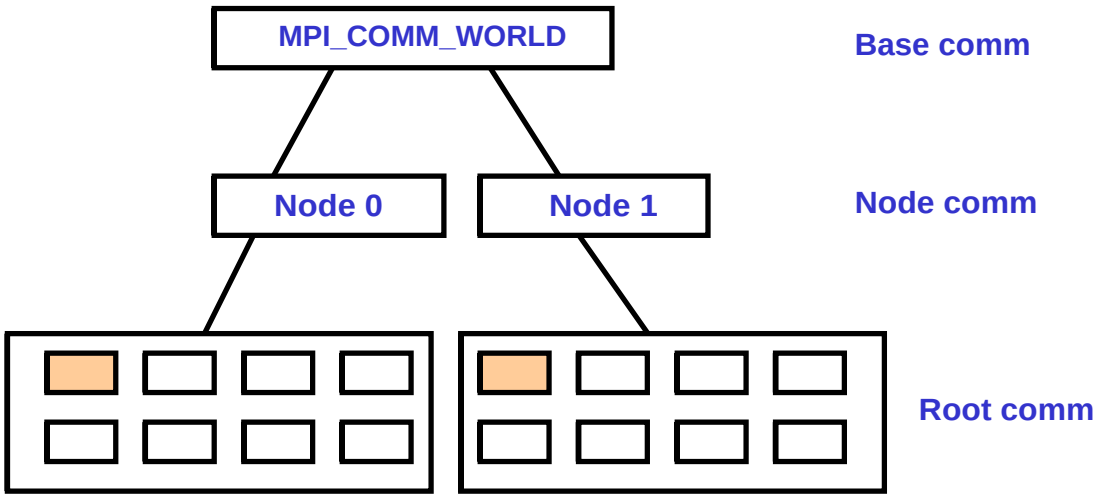

Fig. 8 *fipc_ctxt_world,* shown for 2 nodes with 8 cores each.

Fig. 8 shows the *fipc_ctxt_world* for two nodes with 8 cores each. As mentioned earlier, there are multiple communicators in the *fipc_ctxt_world*. The *base_comm* is similar to MPI_COMM_WORLD and includes all processes that call *fipc_init*. Each

process then determines the node it belongs to by using the *hostname* command. All the processes belonging to the same node create another communicator *node_comm*. For the case in Fig. 8, there will be two communicators for Node 0 and Node 1. Finally, a 'Master' process is selected in each node and they form the *root_comm*. In our example, the two 'Master' processes are highlighted in orange in Fig. 8.

Once the communicator structure is created, the user can then make calls to *fipc_seg_create* specifying the memory block that needs to be created as a SMS. The 'Master' process in each node creates the segment and the rest of the processes use the segment id to attach to the segment created within that SMP node. When all related processing is complete, the SMS can be either deleted through calls to *fipc_seg_destroy*, or they are automatically deleted when a call is made to *fipc_finalized.*

The module also provides calls to *fipc_allreduce* to send the SMS data from one node to another. This only involves the processes in the *root_comm* level. To ensure synchronization while accessing the segments, the user can make use of critical regions through calls to *fipc_critical_start* and *fipc_critical_end*. A barrier is provided through a call to *fipc_barrier*. It should be noted that these synchronization subroutines only act across a single SMP node since only those processes attached to an SMS need to synchronize their access to the SMS. Also, the size of a communicator and the rank of a process within that communicator can be found through calls to *fipc_ctxt_size* and *fipc_ctxt_rank*. For example, from Fig. 8, the size of *base_comm* will be 16, the size of *node_comm* will be 8 for each node and the size of *root_comm* will be 2.

The pros of using SMS are obvious. We can now use a single designated area of memory that can be shared among processes. The use of SMS has a greater benefit in those cases where we have large input data or data that is generated only once throughout the execution of the program and then only read subsequently. In such a case, a critical region or a semaphore is not necessary to access the segment, whereas a write to the segment will require synchronization and thus increase the execution time. The ASCOT code makes use of such data, for example the magnetic field. This data is not changed during execution and makes it an ideal candidate to be used as a SMS. The code makes use of Splines to represent the magnetic field and one can determine the magnetic field strength at a particular point using Spline interpolation.

We did some tests with the cubic spline functions provided by Numerical Recipes (NR) library. The library provides two functions, *'spline'* to calculate the second derivative of the interpolating function and the *'splint'* function to get the value of the interpolated function at a particular point. To calculate the second derivative, we need the input points *x* and the value of the function *y* at each point *x,* given by *y = f(x)*. Since, the value of the input points *x* and *f(x)* are required only once, we can allocate a SMS to store them. The *'spline'* function needs to be called only by process 0 and the output of the function, say *spline_out*, can be used by all the other processes within the same SMP node, as a SMS as well, during interpolation. Our use of SMS was found to work well with the Spline functions from the NR library.

The ASCOT code makes use of the EzSplines library which can also use the SMS in the same way. With our input, Simppa Äkäslompolo was able to write an initial version of the module combining EzSplines and the FIPC module. It needs to be further extended to cater to different kinds of EzSpline objects and multi-dimensional arrays.

6.2.1. Fault safe termination of SMS

One of the issues that a user needs to keep in mind while using SMS is that they are not deleted automatically in the case of a segmentation fault. The operating system might not recognize that such segments exist and can cause the system to run short of memory. We informed HPC-FF support at Jülich and were advised that ParTec has provided a solution to clean up the segments after a job has terminated. However, this will not help in cases where we have a batch script that has multiple calls to *mpiexec*. For example, if the batch script does independent runs of a program with different parameters, then a crash during one of the runs will affect the rest of the runs as well. Due to the persisting SMS, the subsequent program executions might run out of memory. We have included a clean-up mechanism that cleans up these segments after a program terminates abnormally even when the node is in use.

The System V IPC framework provides functions that are used for handling SMS. It also provides methods to implement locks or critical regions that can be used to access the SMS in a synchronized manner, through semaphores. Semaphores are created by one process only and do not require attaching to unlike SMS where each process in the node has to attach to the SMS in order to be able to use it. The FIPC module creates a semaphore initially which is used while accessing the SMS. The clean-up mechanism has to take care of both the semaphore and the SMS in case of an abnormal termination.

The function *shmctl* which is part of the IPC framework, marks an SMS to be destroyed. Passing the *IPC_RMID* parameter along with the SMS id to *shmctl* marks the segment for destruction. However, the segment is only destroyed after all processes have detached from it. The FIPC module has been changed to mark the SMS for destruction once it has been created and all processes have been attached to it. If the program exits normally, then the segments will be deleted automatically. In the case of an abnormal termination, the segments will be deleted if all the processes have detached from it.

There could be instances where the processes do not detach from the SMS after an abnormal termination. Also, semaphores unlike SMS cannot be marked for destruction. In such cases, the segments and semaphores need to be deleted manually using the Linux *ipcrm* command. The FIPC module was also changed to write out the SMS and semaphore ids to a file soon after they are created. The file is then used as an input to a C program that deletes these segments through calls to the *ipcrm* command via the *system()* call. This C program can be executed after the execution of the ASCOT program to ensure that any remaining SMS and semaphores from the program are properly cleaned up.

6.2.2. References

[1] Portable use of Shared Memory Segments, Dr. Ian Bush
www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0701.pdf

6.3. Conclusions on the ASCOT-10 project

The initial working plan envisaged an implementation of a hybrid parallelization model of OpenMP and MPI. However, it became clear that just for reducing memory consumption it was not necessary to implement OpenMP in addition to MPI. The program used large datasets of magnetic field data as input and during the post-processing phase made use of a large amount of memory to store the histogram data for diagnostical purposes. To prevent each core to have its own copy of the magnetic field data, we implemented Shared Memory Segments (SMS) to share the data

among the eight cores within each node on HPC-FF. For the histogram data we implemented a data compression algorithm using a sparse format which significantly reduces memory consumption and also allows the user to use up to seven dimensional histograms. These changes result in a much more efficient memory management of the ASCOT code. As a result it is possible now to simulate more detailed magnetic field configurations which results in more accurate physical simulations.

We would like to thank the project coordinator, Taina Kurki-Suonio, for the good collaboration.

7. Contribution to the JOREK-HR project

JOREK was to be benchmarked as part of the BEUPACK benchmark suite. However, our attempts to port the code to HPC-FF were not successful since the code encountered a deadlock when run on 96 cores or more. We also tested a new version of JOREK which had some issues related to OpenMP fixed and also used a new version of the direct sparse solver PastiX library. Unfortunately, this version also did not run on HPC-FF in the hybrid mode for large problem sizes. A deadlock was encountered with this setup while using ParTec MPI.

To ensure that this was not an issue with the setup on HPC-FF, we tested the code on the IBM Power6 and the AIMS cluster of RZG. Tests were done on 32 MPI tasks with 8 OpenMP threads each, and the job ran to completion on both machines. A difference in these three machines was the MPI library available. IBM Power6 uses the default IBM implementation of MPI, the AIMS machine uses Intel MPI and HPC-FF uses ParTec MPI.

On our request, the HPC-FF support at Jülich installed Intel MPI on HPC-FF for our testing purposes. We then did tests using Intel MPI as well, and these were successful. The code ran through on 8 OMP threads per MPI task even on 512 cores using Intel MPI.

We found that the major difference between the MPI libraries on IBM Power6, Intel MPI and ParTec MPI was that the former two supported MPI_THREAD_MULTIPLE (MTM) level whereas ParTec MPI only supported up to the MPI_THREAD_SERIALIZED (MTS) level. The MTM level is required for the proper functioning of the PastiX library. Hence, we asked HPC-FF support to provide the MTM level of support for ParTec MPI. This requires recompiling the MPI library again with flags that enable the MTM level of support. The code did not run through even with the MTM enabled version of ParTec MPI.

To rule out any program errors, we analyzed the code using the tool Marmot on HPC-FF, which is an MPI correctness checking tool. We were provided a new version of Marmot 2.4.0 which had some bugs in version 2.3.1 fixed, however this was also not functioning correctly. Along with the help of the developer of Marmot, we were able to fix the issue and used the corrected version installed locally on HPC-FF. Analyzing the code with Marmot gave just warnings:

1) Use of MPI_ANY_SOURCE can cause race conditions – usage of MPI_ANY_SOURCE can cause deadlocks and should be avoided if possible.

2) Marmot also reported that the number of elements being sent through MPI_Scatter was less than 0.

These anomalies were reported to Guido Huysmans who is the project coordinator of JOREK.

Since JOREK ran on most machines we tested, and even on HPC_FF with Intel MPI, we came to a conclusion that the problem was in the MTM version of Partec MPI we had available on HPC-FF. For completeness, we also suggested that the code should be looked at by the PastiX developers who could rule out any problems with the library. The library was found to be functioning correctly. The issue was found to be with the MTM level support of ParTec MPI, which they have now fixed, on our request, and is available on HPC-FF as a module that can be loaded by the user. JOREK was finally ported to HPC-FF and Florent Sourbier was able to benchmark it successfully as well.

8. Memory bandwidth on HPC-FF BEUPACK benchmark revisited

We had to revisit the BEUPACK benchmark in the light of additional information on the memory bandwidth that was available on HPC-FF.

On HPC-FF, the compute nodes have a NUMA (Non Uniform Memory Architecture) design. Each node has two Intel Nehalem quad-core processors. The memory is separated into two memory nodes of size 12 GB and each memory node is bound to one of the processors and sockets respectively. Hence, the amount of memory available to one core is approximately 3 GB. The core-MPI task mapping within a node is determined by the MPI execution environment. In the case of HPC-FF, this is determined by the settings Parastation *mpiexec* provides. Due to the NUMA architecture, this mapping can have an effect on the amount of memory and bandwidth available for an application. The BEUPACK benchmark suite has codes that are memory intensive and we chose them to test the mapping strategy used by the *mpiexec* environment. To know how the total available bandwidth impacts the runtime of an application, we need to understand how the cores in an HPC-FF node are attached to a process so that we have an approximate idea of how core mapping can impact memory bandwidth.

HPC-FF also has an environment variable PSI_TPP that can be used to change the distribution of tasks across a node. It is similar to a skip factor which determines how to assign tasks in a compute node of eight cores. The default behavior on HPC-FF is similar to setting PSI_TPP to 1.

Note that it is only important to know the rank of the MPI tasks that are scheduled on each processor. Within a processor, it does not make a difference if the cores are allocated alternately or not since they share the L3 cache and can access the same memory attached to the processor.

The default setting on HPC-FF does not allow a task that is scheduled on Processor 0 to access the memory attached to Processor 1. This can however be changed by setting a value to a variable __PSI_NO_MEMBIND which allows tasks in one processor to access the memory attached to another processor in a node. With this information, we can find out the MPI tasks that are allocated to a processor for varying number of active cores and PSI_TPP values by some simple tests.

We tested a program, on a node on HPC-FF, which uses a maximum of eight active cores and a minimum of two active cores by setting *ppn* (processes per node) to eight and two respectively. The program tries to allocate about 7 GB of memory each to MPI task with rank 0 and another MPI task. If they are both scheduled on the same processor, then the program will terminate with an out of memory exception since a processor has only 12 GB of memory attached to it. Two tasks that are on the same processor will not be able to allocate 14 GB of memory in total with the default setting. This way, we can find out which MPI tasks are allocated on the same processor.

Table 4 shows the mapping of cores to MPI tasks when we have different number of active cores and values of PSI_TPP set, within a node.

From Table 4, when the number of active cores is eight, MPI tasks 0, 1, 2 and 3 are scheduled on Processor 0 and the rest are on Processor 1. We see a difference in the mapping when we have four active cores and PSI_TPP has the default value of 1 or it is set to 2. In the former case all the MPI tasks from 0 to 3 are scheduled on Processor 0 and Processor 1 is idle. When PSI_TPP is set to 2, tasks 0 and 1 are on Processor 0 and tasks 2 and 3 on Processor 1.

Cores	Active cores	PSI_TPP	Core-MPI task mapping Processor 0 Processor 1
8	8	1	0 1 2 3 4 5 6 7
8 / 4	4	1	0 1 2 3
8 / 2	2	1	0 1
8	4	2	0 1 2 3
8	2	4	0 1

Table 4 Mapping of cores to MPI tasks for different number of active cores in a node and PSI_TPP values.

The same difference can be seen when we have two active cores and PSI_TPP is set to 1 or 4. When PSI_TPP is 1, we have both the MPI tasks set on Processor 0 and Processor 1 has no active cores.

The scheduling pattern for two and four active cores with the PSI_TPP set to its default value, implies that the MPI processes have access to less memory than is actually available. In the case of four active cores, one processor is completely idle and its 12 GB memory cannot be accessed by default. The active cores have access to only 3 GB of memory each on Processor 0. In the case of two active cores, they can use only up to 6 GB each even though the other 12 GB of memory attached to Processor 1 are available. One has to use the __PSI_NO_MEMBIND variable to be able to use the memory available on the other processor in both cases. However, when we set PSI_TPP to 2 or 4, the tasks are scheduled on different processors in the node. This automatically means that all 24 GB of available memory is distributed equally among the active cores.

The findings above have been tabulated in Table 5. Applications can benefit from double the amount of memory that is available per core, and also more bandwidth. This can be done by using a suitable number of cores and utilizing the benefits that PSI_TPP can provide to the application.

To use PSI_TPP, the user needs to export the PSI_TPP variable in the shell script. The number of active cores being used is equal to

Number of active cores = Total number of cores / PSI_TPP

For example, to use 512 active cores, the user must specify nodes=128, ppn=8, PSI_TPP=2 and number of tasks=512. This will ensure that the job is run on 512 active cores. The remaining 512 cores are left idling.

ppn	PSI_TPP	Number of active cores per compute node	Memory per core (GB)	Explanation
8	1 (default)	8	3	1 core allocated per task.
8 or 4	1 (default)	4	3	1 core allocated per task.
8 or 2	1 (default)	2	6	2 cores allocated per task.
8	2	4	6	2 cores allocated per task, 2*3GB memory available per core. Doubled bandwidth.
8	4	2	12	4 cores allocated per task.

Table 5 Number of cores per node and memory available per core for different values of PSI_TPP.

The advantage of using PSI_TPP is that the user has a certain degree of freedom while specifying the number of cores during execution. Setting *ppn* will restrict the user to the number of cores requested. For example, the user can set *ppn=8* and set PSI_TPP to two, using only four cores. The user can also run another *mpiexec* within the same batch script that could use eight cores, by setting PSI_TPP to one (default on HPC-FF). However, if *ppn* is set to four, then the user will have to use a new batch script to submit a job of eight cores.

8.1. BEUPACK Benchmark

We have repeated some tests on the BEUPACK benchmark codes to determine the impact of the usage of PSI_TPP on the run times. The tests have been done on 512 cores. The codes JOREK and GYSELA have not been tested as part of this.

The results of using PSI_TPP have been shown in Table 6. The run times are compared to the original benchmark runs where PSI_TPP had its default setting of one.

The timings in Table 6 are as expected. Both MDCASK and ORB5 are Monte Carlo codes where the data locality is high enough to make efficient use of the cache hierarchy. Hence the number of direct requests to access main memory is moderate. Both GENE and GEMR are grid based codes and thus require a large number of direct requests to main memory to sweep through the large grids. The advantage of using PSI_TPP=2 for such codes can be clearly seen as they benefit from a larger memory bandwidth per core.

Users should however note that the percentage gain should be considered carefully before they decide to let every second core run idle in all their jobs. Even though the gain for some codes in Table 6 is obvious, the real benefit in letting every other core idle is only in those cases where the bandwidth gain is more than 100% for a code that already has an ideal scaling curve. If this is not the case, the benefit of using double the number of cores instead of double bandwidth will be higher for the user.

Benchmark	Run time in seconds (PSI_TPP=1)	Run time in seconds (PSI_TPP=2)	%Gain
ORB5	309.34	280.12	9.45
MDCASK – 250 Million atoms	4.80 (per time step)	4.64 (per time step)	3.33
MDCASK – 500 Million atoms	9.26 (per time step)	8.82 (per time step)	4.75
GEMR – ITER case (problem size 4096x2048)	2730.28	1985.62	27.27
GEMR – ITER case (problem size 2048x1024)	682.37	428.7	37.17
GENE	74.87	49.99	33.23

Table 6 Run times on 512 cores with default value PSI_TPP=1 and with PSI_TPP=2.

9. Final report on the MGEDGE project

9.1. Introduction

The multigrid method is a well-known, fast and efficient algorithm to solve many classes of problems including the linear elliptic, nonlinear elliptic, parabolic, hyperbolic, Navier-Stokes equation and Magnetohydrodynamics (MHD). Although the multigrid method is complex to implement, researchers in many areas think of it as an essential algorithm and apply it to their codes because the number of operations of the multigrid method depends on the degree of freedom times the number of levels (*log* of the degree of freedom).

To implement and analyze the multigrid method, we have to consider two main parts of the multigrid algorithm, the smoothing operator and the intergrid transfer operator, separately.

To parallelize the program and to get a good performance, we need to have a good load balance over all cores. In general, the ratio of communication to computation on a coarse level grid is larger than the ratio on a fine level grid. Since the multigrid method works on both coarse and fine grid levels, we need to consider, in detail, the balance between computational work and communication. Usually, the multigrid method requires more work on coarse problems in comparison to other iterative and direct methods. The balance between computation and communication is highly dependent on machine architecture and problem sizes, so we need to determine the level at which we need to stop coarsening according to the number of cores on each machine and problem size.

In this project, we have worked on the efficient implementation of the multigrid method in the GEMZ (Gyrofluid ElectroMagnetic) code of Bruce D. Scott, which solves nonlinear gyrofluid equations for electrons and one or more ion species in tokamak geometry. Starting from an already existing implementation of the multigrid method we amended the intergrid transfer operators and the linear solver. We further continued with detailed adaptation and testing of the implemented multigrid algorithm on the VIP machine at RZG and HPC-FF machine at JSC to finally improve its parallel scalability significantly.

9.2. Implementation of the multigrid method

As part of our efforts, we have assessed the multigrid method to solve the Poisson problem on a rectangular domain with uniform meshes using cell-centered finite differences (CCFD). In focus was the performance of the implemented multigrid algorithm on massively parallel machines with up to many thousands of processors.

	R_0 P_0		R_1 P_1	
highest level	# of iter.	error red.	# of iter.	error red.
6	9	0.0605	9	0.0586
7	9	0.0650	11	0.0939
8	10	0.0795	12	0.1386
9	12	0.1032	13	0.1507
10	13	0.1261	13	0.1539
11	14	0.1514	13	0.1547
12	16	0.1779	13	0.1549
13	17	0.2082	13	0.1550
14	18	0.2444	13	0.1555

Table 7 The number of iterations and average error reduction factor of the multigrid method.

Initially, it was not clear if the zeroth- or first-order intergrid transfer operators would give better performance for the CCFD method. Corresponding tests revealed that in the case of the first-order intergrid transfer operator the average error reduction factor, i.e., the ratio of post-error to pre-error after passing a V-cycle averaged over the number of iterations and the number of iterations did not change with the number of multigrid levels (see Table 7).

Instead the zeroth-order intergrid transfer operator imposes a strong correlation of these quantities to the number of multigrid levels. Thus, we conclude that the first-order intergrid transfer operator should be preferred for larger problems.

We investigated four popular solvers as coarsest problem solvers on the "lowest level"; the Red-Black Gauss-Seidel Relaxation method, sparse direct solver (IBM WSMP), dense direct solver (LAPACK), and Conjugate Gradient Method (CGM). The solution time of each solver was measured for several different problem sizes from 2^8 to 2^{22} Degrees of Freedom (DoF). For our smallest problem size (2^8 DoF), CGM, relaxation, LAPACK(solving) are faster than the other solvers and CGM is the fastest solver for relatively small problem sizes (from 2^{10} to 2^{16} DoF). For larger problem sizes (>2^{16} DoF), multigrid and WSMP back substitution give similar good results (see Fig. 9).

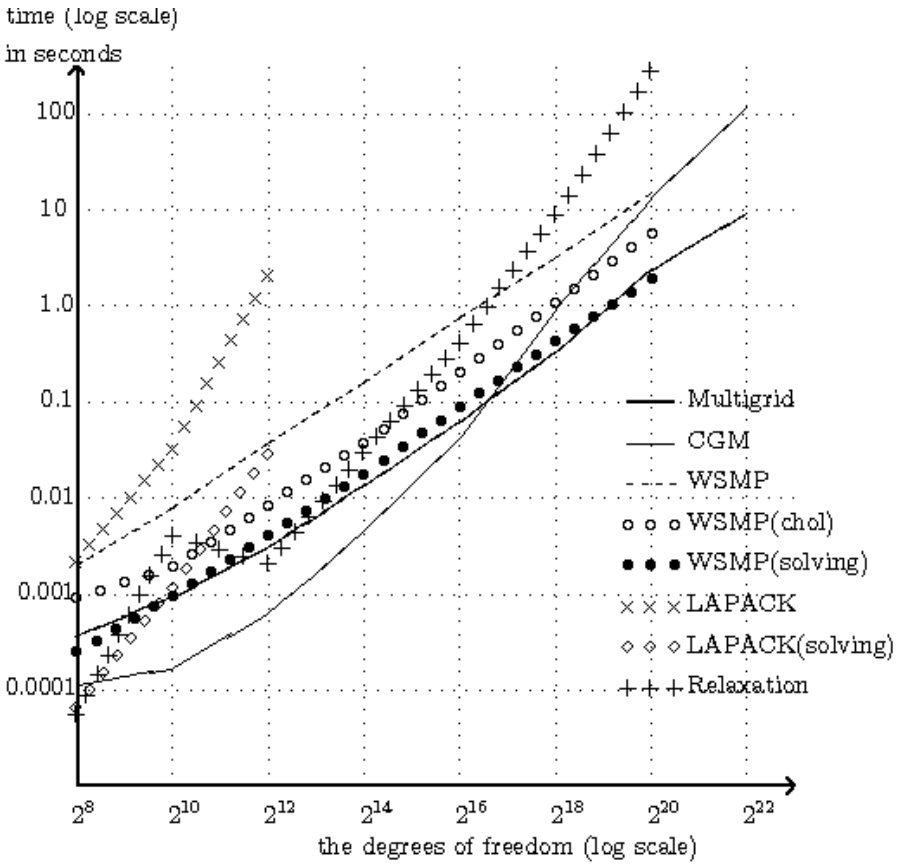

Fig. 9 Solution times of different solvers on a single core of an IBM Power6 757 system.

In addition, we compared the parallel performance of the solvers. The results show that CGM is again the best parallel solver for problem sizes with 2^{10} to 2^{16} DoF. As expected, the optimal number of cores increases with the size of the problem.

Next, we focused on the scaling properties of the parallel WSMP and the multigrid algorithm to solve a problem with 2^{20} DoF. The back substitution of WSMP is better than the multigrid method on small number of cores, less than four cores for VIP and

less than sixteen cores for HPC-FF. For larger numbers of cores, the multigrid method is faster than WSMP due to better scaling properties.

This result suggests that we might get the fastest solution time with one of the following strategies. Either we combine the parallel multigrid method with the parallel CGM solver as “lowest level“ solver, i.e., using the CGM solver on the coarsest grid to get the exact solution. Or we gather the data from all the cores on each core at a certain level (called “gathering level“) and proceed with the single-core version of the multigrid method until we reach the coarsest grid level (see Fig. 10). As “lowest level“ solver we would have again the choice between the CGM, relaxation, and LAPACK/WSMP solver. The second strategy has the benefit that it can enlarge the applicability of the parallel multigrid algorithm for a fixed problem size to very large numbers of cores. In such cases, the degree of freedom at certain levels may be less than the number of cores when approaching to the “lowest level“. Hence, one has to switch to the single-core version of the multigrid method for calculating the ”single core levels” (see Fig. 10).

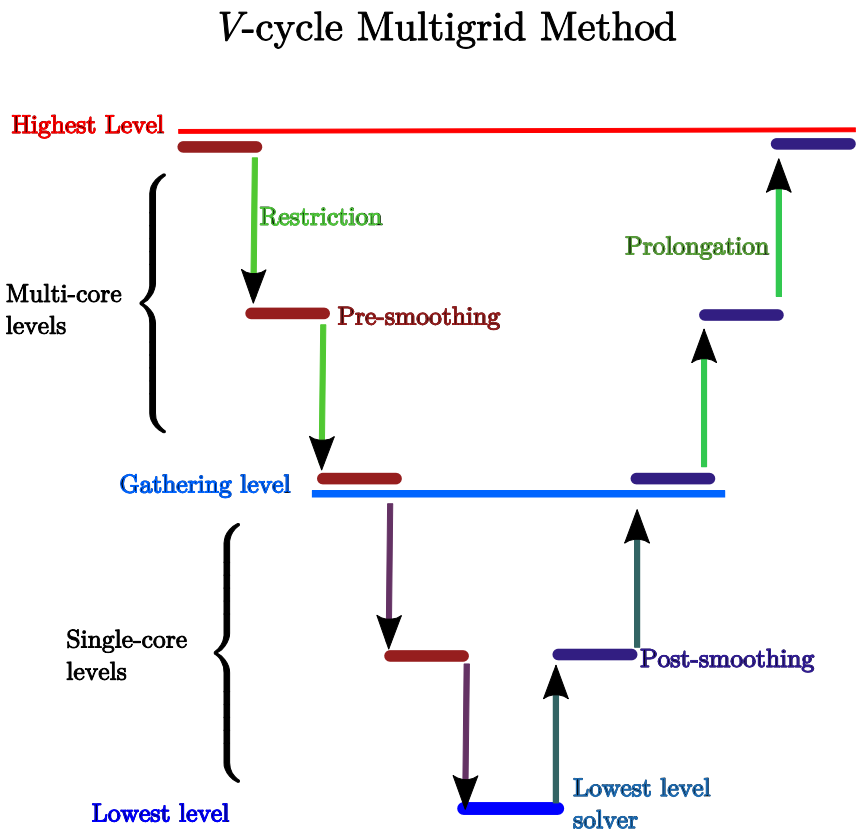

Fig. 10 A schematic view of the V-cycle multigrid method which starts as a parallel multigrid implementation, then after passing the gathering level converts to a single-core multigrid version and finally ends up in one of the possible choices for the lowest level solver.

9.3. Scaling properties of the multigrid method

Detailed tests show that within the uncertainty of the measured execution times the parallel CGM “lowest level“ solver with a “lowest level“ of five gives the best results for a problem size of 2^{24} on HPC-FF up to the maximum chosen number of 512 cores. Thus, only in cases where the number of cores is larger than the number of DoF a single-core multigrid version should be taken into account. A strong scaling of the optimal solver shows a perfect linear speed on up to 512 cores on HPC-FF (see Fig. 11).

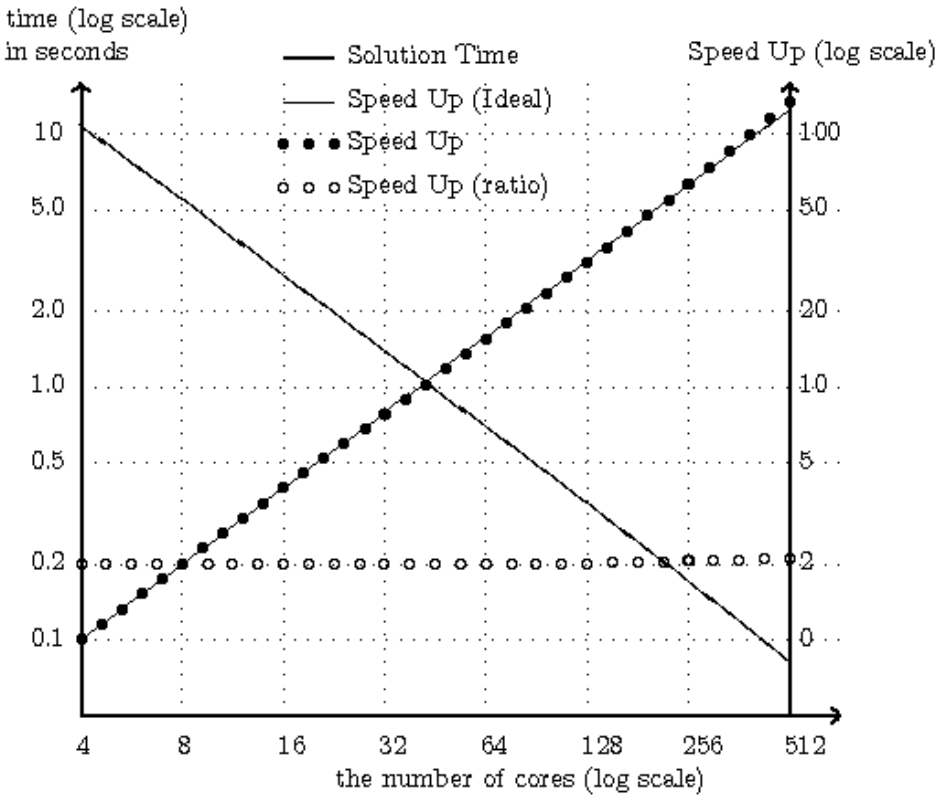

Fig. 11 The solution times, speed up, and speed up ratio of the multigrid method according to the number of cores on HPC-FF to solve a problem with 2^{24} DoF.

Finally, we investigated the weak scaling properties of the optimal solver for four different test cases, i.e., 2^{18} DoF, 2^{19} DoF, 2^{20} DoF, and 2^{21} DoF per core with a weak scaling from one core to 2048 cores on HPC-FF. The multigrid method has very good weak scaling properties. The larger the problem size per core the better it scales. For the two largest test cases with 2^{20} DoF and 2^{21} DoF per core, the execution time increases by less than 5%. This is remarkable as the scaling spans an increase of the core number by a factor of 1024 (see Fig. 12).

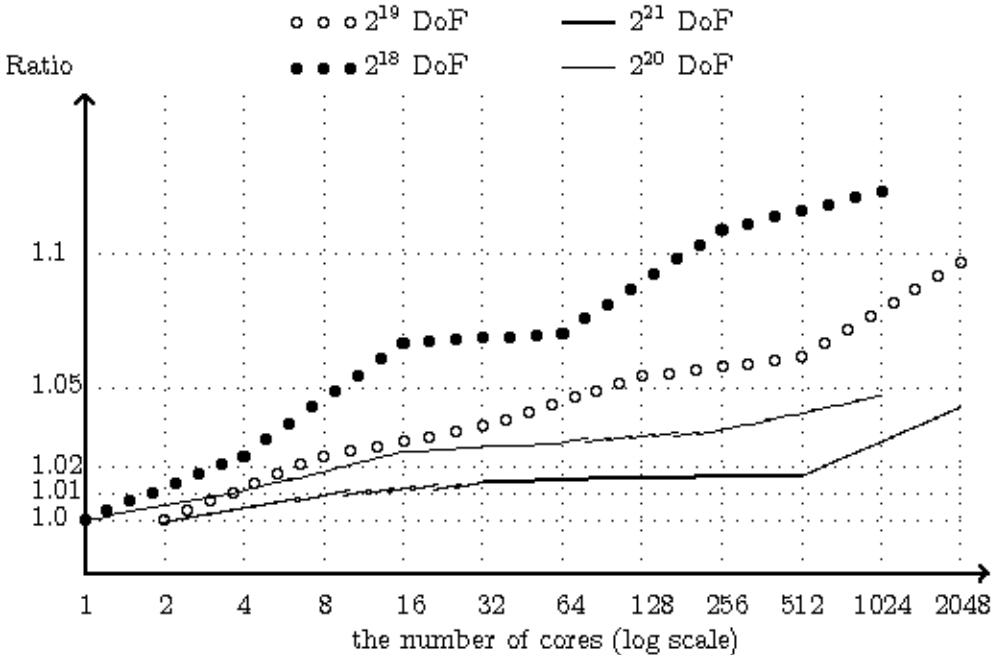

Fig. 12 The relative solution times according to the number of cores on HPC-FF to solve problems with a fixed DoF per core (weak scaling).

9.4. Conclusions on the MGEDGE project

Over all, we proved that our implementation of the multigrid method with the conjugated gradient method as a “lowest level“ solver and with first-order intergrid transfer operators has very good strong and weak scaling properties. Thus, it is suitable for usage on massively parallel machines like HPC-FF. For details please see the HLST report *“Parallelization of the Multigrid Method on High Performance Computers”* which has been published as IPP report 5/123.

We have discussed our results with the project coordinator, Bruce D. Scott, and given him feedback about the changes we did in his 2-dim multigrid test code version. Next step, the implementation of the new multigrid implementation into the production version of GEMZ will be done by himself. In addition, we also had fruitful discussions with Bruce D. Scott about the multigrid method on a triangle mesh with finite volume discretization.

10. Report on the KinSOL2D project

10.1. Introduction

The Particle-in-Cell (PIC) code BIT1 is restricted so far to 1D3V plasma and 2D3V neutral particle modeling with a reasonable scaling up to 1000 and more processors. Hence, ongoing work is focused on enhancement of the code to 2D3V plasma simulations of the Scrape-Off-Layer (SOL). The increase of the dimensionality of the code to 2D or even 3D seems to be straight forward. However, the Poisson solver in 2D has been identified as a bottleneck for the scaling properties. It is mandatory that also this part of the code scales to very high processor numbers to maintain the good scaling property of the whole code. So the work plan is to develop a good scaling Poisson solver in 2D. Possible candidates as solvers are a multigrid solver or, depending on the type of the matrix, a preconditioned Conjugated Gradient (CG) method and Generalized Minimal Residual method (GMRES), respectively. A combination of both is also thinkable where the multigrid method is used as a preconditioner for either the CG or the GMRES method.

The GMRES method has to be used for non-symmetric or non-positive definite systems which can arise e.g. through the boundary condition treatment. In general, the preconditioned system of a symmetric system is not symmetric for the same inner product. However under certain conditions such a system can be symmetric in a different inner product (*A*-inner product or energy inner product) and the less costly CG method can be used.

The multigrid method is a well-known, fast and efficient algorithm to solve many classes of problems including the linear elliptic, nonlinear elliptic, parabolic, hyperbolic, Navier-Stokes equation, and Magnetohydrodynamics (MHD). Although complex to implement, researchers in many areas think of it as an essential algorithm and apply it to their codes because the complexity of the multigrid method is only *N log(N)* , where N is the degrees of freedom (DoF).

To implement and analyze the multigrid method, we have to consider two main parts of the multigrid algorithm separately: the intergrid transfer operators and the smoothing operator. The intergrid transfer operators depend on the discretization scheme and are highly related with the discretization of the matrix. The smoothing operator will be implemented according to the matrix-vector multiplication. So we have to determine the appropriate discretization method which includes the generation of the matrix and an efficient implementation of the matrix-vector multiplications.

When a multigrid solver converges, it usually converges very fast which is the case on most of the problems. However, the multigrid method as a solver does not guarantee convergence. In contrast, iterative Krylov subspace methods which include the CG and the GMRES method guarantee convergence and can be further improved by preconditioners to speed up the convergence rate. The multigrid method is also well-known to act as a very efficient preconditioner. The preconditioned CG method can be used only for symmetric and positive definite problems and has to use the A-norm instead of the L^2-norm. The preconditioned GMRES method works for non-symmetric or non-positive definite problems, but needs more working memory. To reduce the working memory in the GMRES method the Restart GMRES algorithm is an option which does not guarantee convergence, but converges for most of the problems. In the following we will investigate the multigrid method itself as a solver or as a multigrid preconditioner for a Parallel GMRES (PGMRES) solver.

10.2. Model problem

We consider the second order elliptic partial differential equation with the coefficient ε(x,y) being defined on a rectangular domain with an internal conducting structure as in Fig. 13. The boundary conditions are as follows: Dirichlet zero boundary condition for the outer boundary, Neumann zero boundary condition for the inner empty surface and time-dependent Dirichlet boundary condition for the internal conductor.

The finite difference method is chosen as discretization method. In addition, a structured mesh with ε(x,y) being defined on each cell boundary is used. The coefficient ε(x,y) is time dependent and will change for each time step of the simulation.

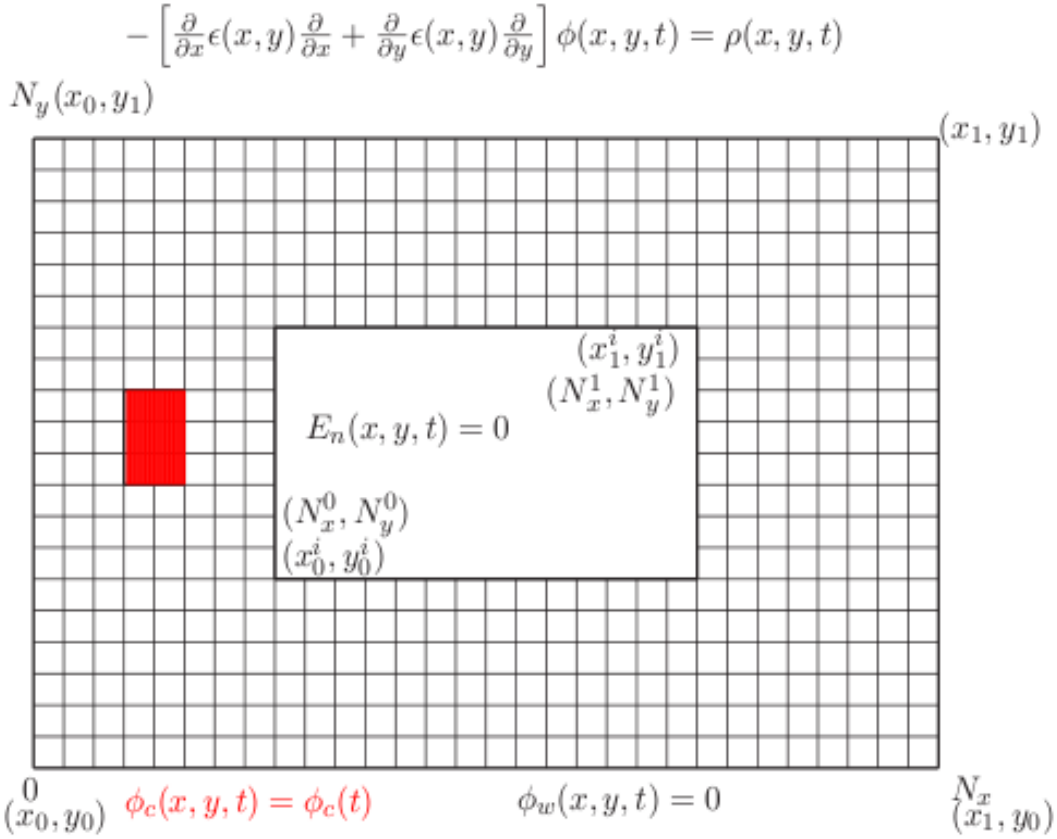

Fig. 13 Rectangular domain with internal conducting structure [■: internal structure (conductor)]

As the model problem is quite complex it is advantageous to split it first into subproblems which can be solved separately. After successful treatment of the subproblems they can be used as building blocks to assemble an algorithm for the full problem. Hence as a first step, we focus on the convergence property of the solver for an internal conducting structure which can be arbitrarily shifted relative to the coarsest mesh of the multigrid method. For the multigrid method this is a challenging problem as the corners of the conducting structure and inner empty space could lead to a poor convergence rate of the multigrid method.

10.3. Multigrid software framework

To prevent starting each new multigrid project from scratch, we have started to develop a multigrid software framework. So far a preconditioned CG, a preconditioned GMRES and a multigrid solver have been implemented. As an efficient preconditioner the multigrid method can be used. The discretization method has only an impact on the matrix-vector multiplication and the construction of the

matrix. Special versions of the vector multiplication for finite volumes, finite elements and finite differences have been provided. It is the vector multiplication where the parallelism is introduced into the framework. Corresponding tests have been done to prove the correctness of the implementation. For the discretization with finite difference considered here with a Neumann boundary condition, the generated matrix is non symmetric, so the CG method cannot be used. Instead only the preconditioned GMRES method and the multigrid method can be used.

The implementation of the parallel matrix-vector multiplication and the smoothing operators, in our case Gauss-Seidel and Jacobi, is highly dependent on the domain handling of the parallelization concept. In our case we discretize the whole rectangular domain in each direction with a uniform mesh and divide the rectangular domain in nx by ny small rectangular sub-domains which are handled by one processor each. If a rectangle sub-domain is in the inner empty space, we do not need to handle this domain and thus do not assign a processor to such a sub-domain. As a result a certain fraction of the processors will idle. In the future this can be further refined but at the moment it seems to be justified by the fact that the inner empty space is much smaller than the reaming area of the rectangular domain. In case of the internal conducting structure no exception is made so that all sub-domains are assigned to a processor nevertheless they include parts of the internal conducting structure or not.

10.4. Tests with inner conducting structure

We have tested the correct discretization of the elliptic problem on a rectangular domain with an internal conducting structure and an inner empty space by comparing the converged numerical solution with the exact solution. To construct an exact solution for a zero Neumann boundary condition on the boundary of the inner empty space, we chose the following sine function f(x,y):

$$\frac{1}{2\pi^2}\sin^2\pi x \sin^2\pi y$$

The Dirichlet boundary condition of the internal conductor is given by the values f(x,y) at the boundary of the conductor. For simplicity we chose for the mesh sizes h=hx=hy. The according L^2 discretization errors are listed in Table 8. It can be clearly seen that the error converges by second order ($O(h^2)$).

level	L^2 error	h	Error ratio
7	0.00005786	0.031250000	–
8	0.00001440	0.015625000	4.0182
9	0.00000359	0.007812500	4.0082
10	0.00000090	0.003906250	4.0039
11	0.00000022	0.001953125	3.9996
12	0.00000006	0.000976562	3.9984

Table 8 The discretization L^2 error and the error ratio between succeeding refinements.

Next we test the multigrid solver and the PGMRES method in combination with the multigrid preconditioner. We have implemented the first order intergrid transfer operator and tested it. We use the Jacobi iteration and the local Gauss-Seidel iteration as smoothing operators. These two smoothing operators are relatively simple and well analyzed. The Jacobi iteration does not have a good performance, but does not depend on the number of processors. This is beneficial when testing the parallelized multigrid method because results do not depend on the number of

processors being used. In contrast the local Gauss-Seidel iteration has a good performance, but depends slightly on the number of processors.

As a test case, we chose a 4 by 4 rectangular domain with a 2 by 2 inner empty space and a fixed finest mesh with h=hx=hy=0.001953125 and several different coarse meshes for the V-cycle. We always chose the inner empty space in such a way that it is aligned with the coarsest mesh. In contrast the inner conducting area can have an arbitrary shift relative to the coarsest mesh.

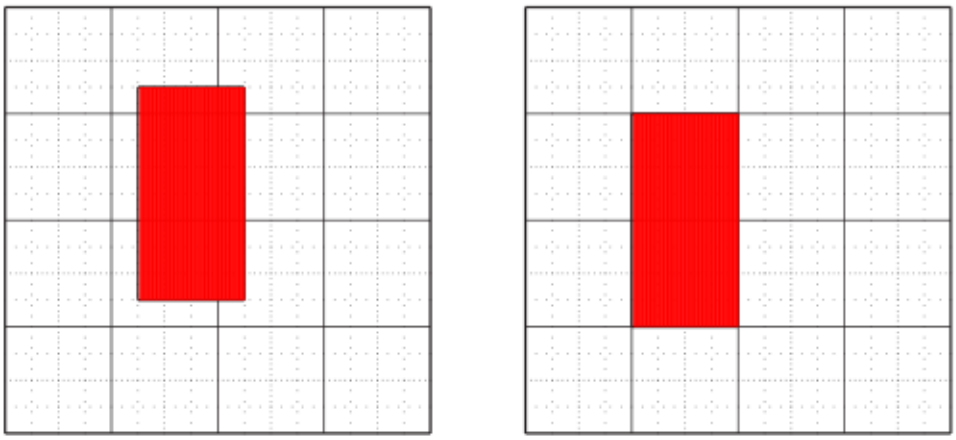

Fig. 14 The positioning of the internal conducting area [■: internal structure (conductor), line: coarsest mesh, dotted line: finest mesh]

We sketch two different configurations of the internal conductor area in Fig. 14, one does not match with the coarsest mesh (left) and the other one matches the coarsest mesh (right). The average error reduction factors of the multigrid method, i.e., the ratio of post-error to pre-error after passing a V-cycle averaged over the number of iterations, are listed in Table 9 according to the selected smoothing operator and the number of levels of the V-cycle. The solution on the coarsest mesh is computed by using the GMRES method.

The results show that the multigrid method as a solver and the PGMRES method with a multigrid preconditioner have a very good performance if the internal conducting area matches with the coarse meshes. Otherwise the convergence rate is significantly reduced. In some cases this can lead to a non converging multigrid solver result. Nevertheless, the corners of the conducting structure and inner empty space do not seem to be a problem.

	Jacobi iteration		Gauss-Seidel	
levels	MG	PGMRES	MG	PGMRES
3	0.434595	0.153167	0.321513	0.096483
4	0.350831	0.145256	0.295646	0.090466
5	0.750798	0.197123	0.510491	0.129561
6	**	0.207959	0.835621	0.137452

(a) Unmatched internal conductor area. Note, "**" marks a non converged result.

	Jacobi iteration		Gauss-Seidel	
levels	MG	PGMRES	MG	PGMRES
4	0.268014	0.090191	0.131479	0.042970
5	0.290220	0.097606	0.144441	0.049281
6	0.307084	0.109197	0.157332	0.050775
7	0.323019	0.112501	0.168422	0.058537
8	0.318262	0.121409	0.160810	0.066247

(b) Matched internal conductor area.

Table 9 The average error reduction factor of the multigrid method as a solver and preconditioned GMRES with the multigrid preconditioner.

Next it is interesting to see how the GMRES method behaves without using a preconditioner. For this purpose we compared the solution times of the multigrid solver and the preconditioned PGMRES solver with the GMRES solver for two different mesh spacings of h_1 = 0.0078125 and h_2 = 0.00390625 on a parallel 24 cores run on HPC-FF. In this comparison we consider the case that the internal conducting area matches with the coarsest mesh. For the multigrid method we use the local Gauss-Seidel smoother together with a total number of six levels. The corresponding solving times are listed in Table 10 together with the number of Degree of Freedoms (DoF). It can be clearly seen how the multigrid preconditioner speeds up the GMRES method significantly.

h	# DoF	MG	PGMRES	GMRES
0.0078125	194281	0.449	0.246	17.6
0.00390625	778323	0.901	0.535	216.8

Table 10 The solving times in seconds on 24 cores on HPC-FF at JSC.

10.5. Future plans

The method as it has been tested so far has still some limitations. Good convergence of the multigrid and PGMRES solvers is achieved for an inner empty space and internal conducting area which matches the coarsest grid. Hence, it will be of interest how these limitations can be overcome. Finally, the method will have to prove in a weak and strong scaling test how its scaling properties are on the HPC-FF machine. In addition, one has to see how the method competes with the parallel direct sparse solver from IBM WSMP.