EUROfusion

EUROFUSION WPISA-REP(16) 16103

R Hatzky et al.

HLST Core Team Report 2009

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 2009

Contents

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
BEUPACK/JOREK-HR	Nitya Hariharan	finished/running
GYGLES	Roman Hatzky	finished
OPTGS2	Nicolay Hammer	finished
MGEDGE	Kab Seok Kang	running
GYNVIZ	Matthieu Haefele	proposed

Table 1 Projects mapped to the HLST core team members

Nitya Hariharan started with the JOREK-HR project as the employment of the selected HLST staff member, Florent Sourbier, was delayed by administrative issues. Hence, she was predestined to take charge of the BEUPACK project which includes the JOREK code in its benchmark suit.
The objectives of the project have been achieved. For each code in the benchmark suite a report has been sent to the project coordinator, Jacques David (see Appendix 2.2). A detailed report on the project BEUPACK is given in Sec. 2.
Florent Sourbier has taken over the JOREK-HR project but Nitya Hariharan will give further support for the project about some issues concerning the native MPI version on HPC-FF which could not be solved with the vendor ParTec in the past.

Roman Hatzky worked on the GYGLES project. A memory reduction of the dominating matrix part of the code could be achieved by a factor of 20. Hence, simulations can now run on HPC-FF which would not have fit into the memory before. In addition, the scaling could be improved from 128 to 512 cores. The new code version has been delivered to the principal investigator, Alexey Mishchenko, and is running in production on HPC-FF.
The objectives of the project have been achieved. Part of the suggested work was done in the ORBIS project by Trach-Minh Tran. A detailed report on the project GYGLES is given in Sec. 3.

Nicolay Hammer worked on the OPTGS2 project. In the focus was a systematic study of Discontinuous Galerkin (DG) methods to find an alternative to resolve the performance bottleneck of the gyrokinetics code GS2, which arises from the implicit finite difference scheme used in the solver. Many different schemes have been implemented in a testbed. The analytical fundamentals of the DG method and the numerical results are expressed in a technical report of 24 pages called "*Combining Runge-Kutta discontinuous Galerkin methods with various limiting methods*". The report is available via the HLST web site (URL: http://www.efda-hlst.eu/tutorials). The report and the corresponding DG algorithm testbed have been passed to the project coordinator, Wayne Arter.
The objectives of the project have been achieved. A detailed report on the project OPTGS2 is given in Sec. 4.

Kab Seok Kang recently started to work on the MGEDGE project. Several meetings with the project coordinator, Bruce D. Scott, took place. A test problem has been set up to study the behaviour of the multigrid algorithm in detail.

Matthieu Haefele launched a new project, called GYNVIZ. It is the result of a request, for graphical support, from several European gyrokinetic turbulence simulation groups in the first call for the use of HLST resources. Instead of targeting on individual solutions, his proposal offers a general solution by involving the principal investigators of the major European gyrokinetic simulation codes. It addresses the general issue of data post-processing and visualization for the gyrokinetic codes' outputs. Further project details can be found in Sec. 5.
A proposal has been submitted to the board as part of the recently closed call for the use of HLST resources.

1.2. Further tasks and activities of the core team

1.2.1. Dissemination

The web site HLST – High Level Support Team has been set up by Roman Hatzky together with Andreas Schmidt from RZG. The web site informs about the goal and current status of the project EFDA-HPC. In addition, it is used as platform to announce news concerning the project. An internal, password protected domain has been set up to coordinate both parts of HLST (core + staff). For this purpose a wiki has been installed. In addition, a version control system (SVN) can be used as a repository for the codes of the different HLST projects.

In collaboration with the OPTGS2 project coordinator, W. Arter, an abstract for the 21st International Conference on Numerical Simulation of Plasmas 2009 (ICNSP 09) was submitted (http://icnsp09.ist.utl.pt/code/preview.php?abs_id=32). The talk was given by W. Arter on 8th October at the ICNSP 09 in Lisbon.

1.2.2. Internal training

Nicolay Hammer and Nitya Hariharan have attended:
Introduction to the programming and usage of the supercomputing resources in Jülich, Jülich, 10th – 11th August, 2009.

IPP Summer University on Plasma Physics and Fusion research, Greifswald, 21st – 25th September, 2009.

Nicolay Hammer has attended at HLRS:
Advanced topics in parallel programming 2009, Stuttgart, 15th – 16th October, 2009.

Nitya Hariharan has attended at Technische Universität München the lecture:
Scientific Computing I – winter, 2009.

1.2.3. Workshops & conferences

Matthieu Haefele has attended:
EUFORIA general meeting, Strasbourg, 25th – 27th November, 2009.

Roman Hatzky has attended:

- *21st International Conference on Numerical Simulation of Plasmas, Lisbon*, 6th – 9th October, 2009.
- *IPP Theory Meeting, Sellin*, 16th – 20th November, 2009.

1.3. Recommendations for the year 2010

The call for the use of High Level Support Team resources was closed on 31st December, 2009. The core team leader has written a suggestion on how to distribute

the work load over the HLST members. The corresponding spreadsheet has been 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 the core team members will, in agreement with the HLST coordinator, continue to work on the projects of 2009 listed in Table 1. In the case of the projects GYGLES and OPTGS2 supplementary work will be done.

2. Report on BEUPACK project

2.1. BEUPACK benchmark suite

The intent is to prepare the fusion codes package for use as basis benchmark codes for IFERC procurement. To do so requires getting reference data on HPC-FF as a future reference system, and then prepare packages of codes and tests case for further use.
There were six codes in total, ORB5, GENE, GEMR, JOREK, MDCASK and GYSELA that were to be benchmarked as part of the BEUPACK benchmark suite. Since HPC-FF is a relatively new machine, we also hoped to gain new insights into the scaling behaviour of the machine concerning a representative sample of codes.

We have, in all benchmark cases, tried to set problem sizes large enough to occupy the maximum possible memory per core, which is around 3 GB. This ensures that the system is under enough load so as to give reliable statistics.

A summary of the benchmarks and some of the relevant findings are given in the following sections.

2.1.1. ORB5

The ORB5 code was provided by Alberto Bottino who was the coordinator for the benchmark. Test cases for both Strong and Weak scaling were also provided. The benchmark was to be done from 512 cores onwards for Strong scaling and 256 cores onwards for Weak scaling, up to a maximum of 4096 cores.

We found that the code was not able to run on a large number of cores, e.g., 4096 cores. On investigating further, this was found to be due to the amount of memory available per core on HPC-FF. Each MPI connection on HPC-FF requires about 0.55 MB of memory. For 4096 cores, this is about 2.2 GB per core. Taking into account the memory occupied by the operating system this leaves about 500 MB for the application which is not sufficient.

On HPC-FF, an environment variable PSP_ONDEMAND controls the allocation of MPI buffers. By default it is set to 0. This means that all relevant MPI buffers are allocated before the execution of the program. Doing so is not very useful if the application does not use all of the already allocated buffers. Setting PSP_ONDEMAND to 1 allocates buffers dynamically during runtime as and when necessary, leaving more memory available to the application.

We ran ORB5 with PSP_ONDEMAND set to 1, and these runs were successful even on 4096 cores. We were able to do Strong scaling and Weak scaling on HPC-FF without any other issues. The problem size given for Strong scaling was found to be the maximum possible problem size on 512 cores. Hence, to get comparable results, we quadrupled the original problem size for Weak scaling on 512 cores to set it to be the same as that on 512 cores for Strong scaling. The problem sizes on 1024 cores onwards were also quadrupled accordingly. The report given to Alberto Bottino is given in the Appendix 2.2.1.

2.1.2. GENE

The GENE code was provided by Tilman Dannert who was the coordinator for the benchmark. Test cases for Strong and Weak scaling were also given. The

benchmark was to be done from 512 cores onwards, for both Strong and Weak scaling, up to a maximum of 4096.

During some test runs for Strong scaling on HPC-FF, we found that the run times with PSP_ONDEMAND set to 0 were about twice as high when compared to PSP_ONDEMAND set to 1. Since dynamic allocation of buffers, if done after the call to MPI_INIT, should ideally increase the run time, the results were not as expected. It was suggested by the MPI vendor ParTec that this could be due to the use of MPI_ANY_SOURCE in the MPI_Send calls. When PSP_ONDEMAND is set to 0, the communications library has to enquire a long list of receive buffers which adversely impacts the run time. However, when PSP_ONDEMAND is set to 1, the library has to enquire a relatively lesser number of buffers which explains the better performance. The code was then changed to exclude the use of MPI_ANY_SOURCE. Repeating the test runs confirmed what ParTec had suggested.

It was known that setting PSP_ONDEMAND to 1 can cause problems with applications that have an all-to-all communication. However, we did the benchmark by setting PSP_ONDEMAND to 1, due to the significant amount of total memory that GENE requires, and did not encounter any issues. We found that GENE uses all-to-all communications, although not on all processors, but only on split MPI communicators. This explained the successful execution of the benchmark cases on HPC-FF. The report given to Tilman Dannert is given in the Appendix 2.2.2.

2.1.3. GEMR

The GEMR code was provided by Bruce Scott who was the coordinator for the benchmark. Strong scaling, from 512 cores to 4096 cores, was done only for ITER and two cases with varying grid sizes were provided. Weak scaling was done for AUG, JET and ITER cases from 256 cores to 4096 cores.

We did not find any system related issues while benchmarking GEMR. The report given to Bruce Scott is given in the Appendix 2.2.3

2.1.4. JOREK

The JOREK code was provided by Guido Huysmans who was the coordinator for the benchmark. Of all the benchmark codes, JOREK is the only one that uses a hybrid model with both MPI and OpenMP.

We were not able to run JOREK on more than 96 processors without encountering a deadlock. To investigate the cause of the deadlock and to rule out any issues with the code itself we ported the code to the IBM Power 6 and SUN Linux cluster “AIMS” at the Rechenzentrum Garching (RZG). JOREK also uses a number of external libraries which could affect the application and this made it necessary to run tests on other machines.

We found that JOREK and a library that it uses, PaStiX, both required a MPI Thread level support of MPI_THREAD_MULTIPLE (MTM) which was not provided by the native ParTec MPI implementation available on HPC-FF. ParTec MPI only provides support up to MPI_THREAD_SERIALIZED (MTS) where multiple threads can call MPI but only one at a time. MTM on the other hand, allows multiple threads to make MPI calls at the same time, which was essential to the proper functioning of a hybrid code like JOREK. Hence, we requested HPC-FF support to install Intel MPI. Repeated tests using Intel MPI were successful. We were also able to get a new version of ParTec MPI with MTM support enabled installed on HPC-FF, but JOREK did not run through with this version.

We have some open queries related to the MTM version of ParTec MPI and the PaStiX library which have been directed to ParTec. Guido Huysmans has been given a report on our work; a copy is given in the Appendix 2.2.4.

2.1.5. MDCASK

The MDCASK code was provided by Maria Caturla who was the coordinator for the benchmark. Test cases for Strong and Weak scaling were also given. The benchmark was to be done from 512 cores onwards, for both Strong and Weak scaling, up to a maximum of 4096 cores.

On HPC-FF, the code timed out when run on more than 512 cores, using ParTec MPI. It was found that running the code with PSP_ONDEMAND set to 1, which allocates MPI buffers dynamically during run time, could cause this problem if a code uses all-to-all communications. MDCASK uses a considerable amount of hand coded all-to-all communication; this, could effectively create the same scenario as a direct call to the MPI all-to-all routines and stall the code. Tests done with PSP_ONDEMAND set to 0 on 1024 cores for $500 \cdot 10^6$ atoms were successful. However, it was also known that the amount of memory occupied by ParTec MPI when PSP_ONDEMAND is set to 0 can become large enough to cause memory issues for large problem sizes. Since we had the Intel MPI installed while investigating JOREK, we were able to use it to verify if it worked with MDCASK. All test runs were successful, even on 4096 cores.

We also found that the code could not be run on problem sizes larger than approximately $2 \cdot 10^9$ atoms. Exceeding this number also exceeds the maximum value a signed 32 bit integer can hold, i.e., $2^{31}-1 = 2147483647$. This in turn causes an overflow in many places of the code and results in error during execution. Even for smaller numbers of atoms we found the same overflow issue already present with the calculation of the Average CPU time per time step per atom. The code was changed accordingly and sent to Maria Caturla. The report given to her is given in the Appendix 2.2.5.

2.1.6. GYSELA

The benchmark for the GYSELA code was done by Virginie Grandgirard. We have plotted the graphs to be consistent to the plots for the other benchmarks. The report is given in the Appendix 2.2.6.

2.2. Appendix of BEUPACK report

2.2.1. ORB5 Code - Benchmark results on HPC-FF

The benchmark for the ORB5 code has been done on HPC-FF up to a maximum of 4096 cores. The results of Strong and Weak scaling have been given.

2.2.1.1. Strong scaling

The benchmark has been done on 512 cores up to 4096 cores. The speedup and efficiency have been calculated with respect to the timings obtained for 512 cores. Multiple runs have been done to ensure a good representation of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 1, and the speedup in each case is given in Fig. 2.

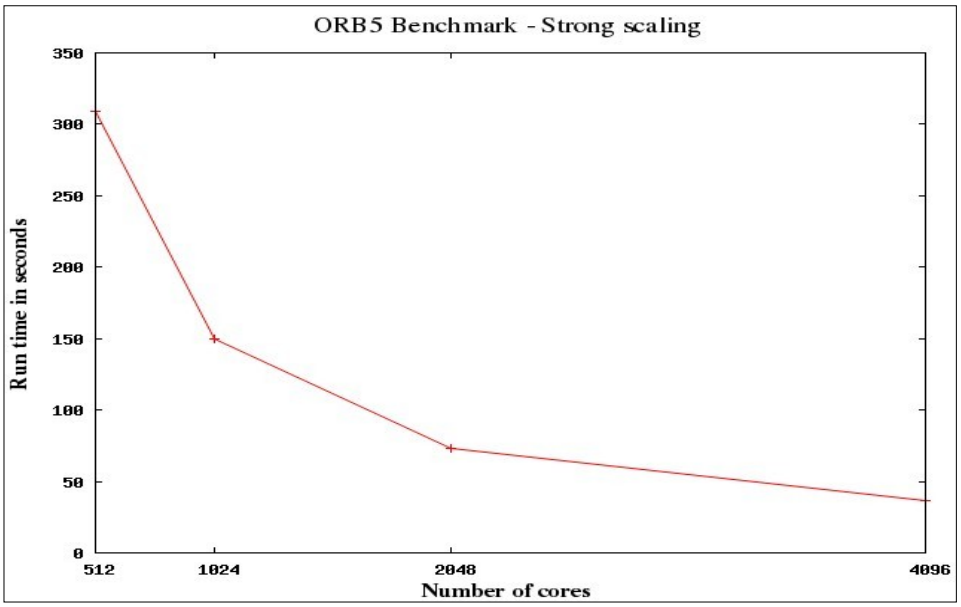


Fig. 1 Benchmark results for Strong scaling, Run time in seconds vs. Number of cores

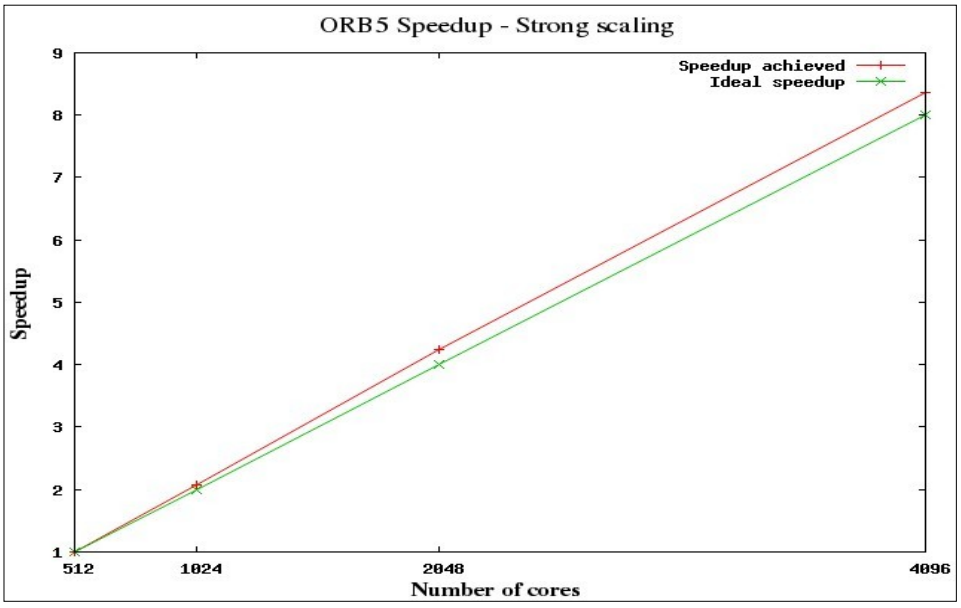


Fig. 2 Speedup for Strong scaling

The number of ions for Strong scaling was initially set to $2048 \cdot 10^6$ for all test cases. To ensure that the maximum possible problem size was being used for benchmarking, this number was doubled to $4096 \cdot 10^6$. This did not result in successful runs on 512 cores due to memory issues. The maximum number of ions suitable for Strong scaling was hence found to be $2048 \cdot 10^6$.

2.2.1.2. Weak scaling

The benchmark has been done on 256 cores up to 4096 cores. The speedup and efficiency have been calculated with respect to the timings obtained for 256 cores. Multiple runs have been done to ensure a good representation of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 3 and the efficiency in each case is given in Fig. 4.

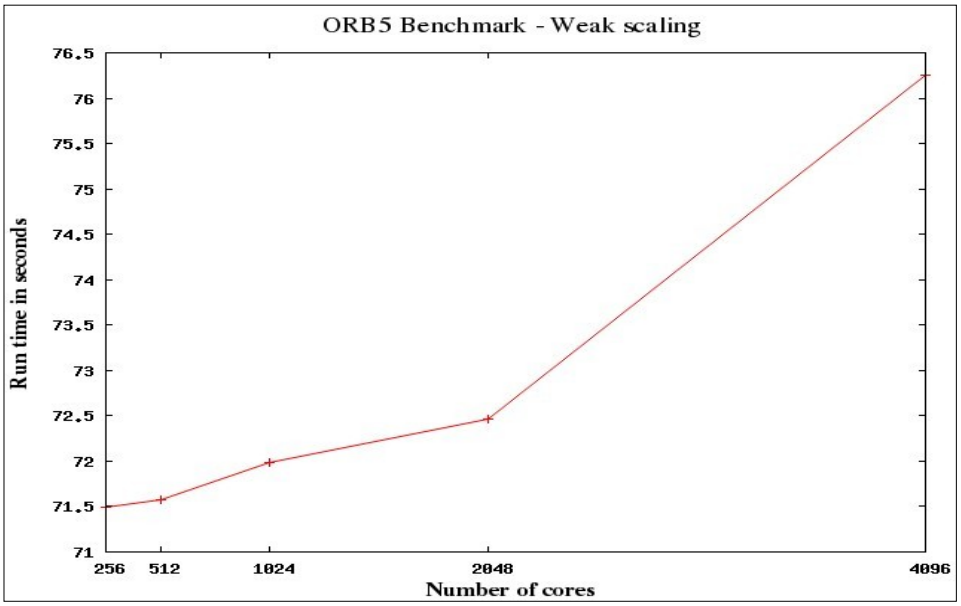


Fig. 3 Benchmark results for Weak scaling, Run time in seconds vs. Number of cores

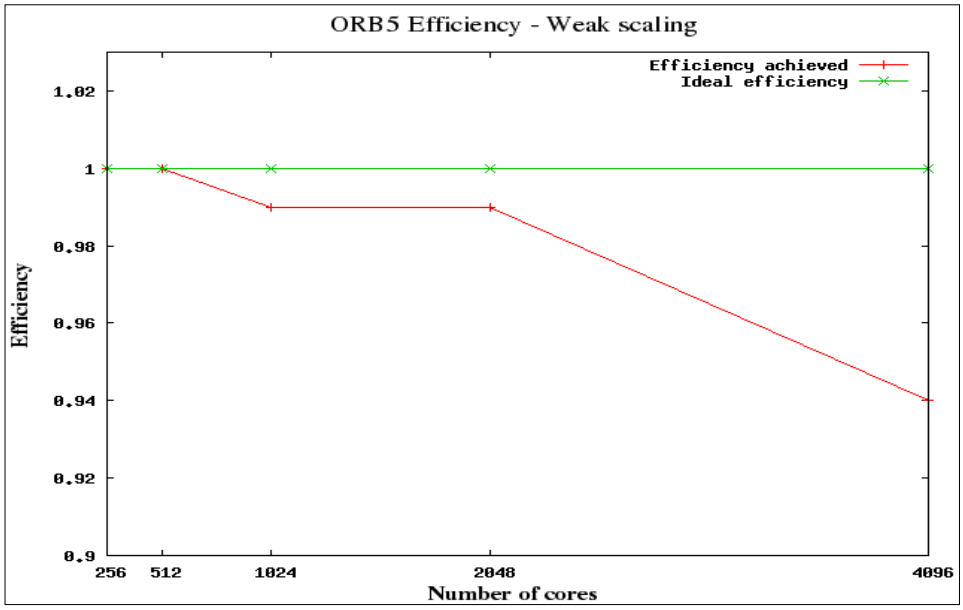


Fig. 4 Efficiency for Weak scaling

Initially, the number of ions for Weak scaling was set to $256 \cdot 10^6$ for 256 cores and then increased by a factor of 2 for twice the number of cores, $512 \cdot 10^6$ for 512 cores etc. Then, the number of ions was quadrupled for each test case. This was done:

- To ensure that the maximum possible problem size was being used.
- To compare it with a similar test case in Strong scaling, e.g., $2048 \cdot 10^6$ ions for 512 cores.

The corresponding benchmark results are given in Fig. 5. The efficiency is given in Fig. 6.

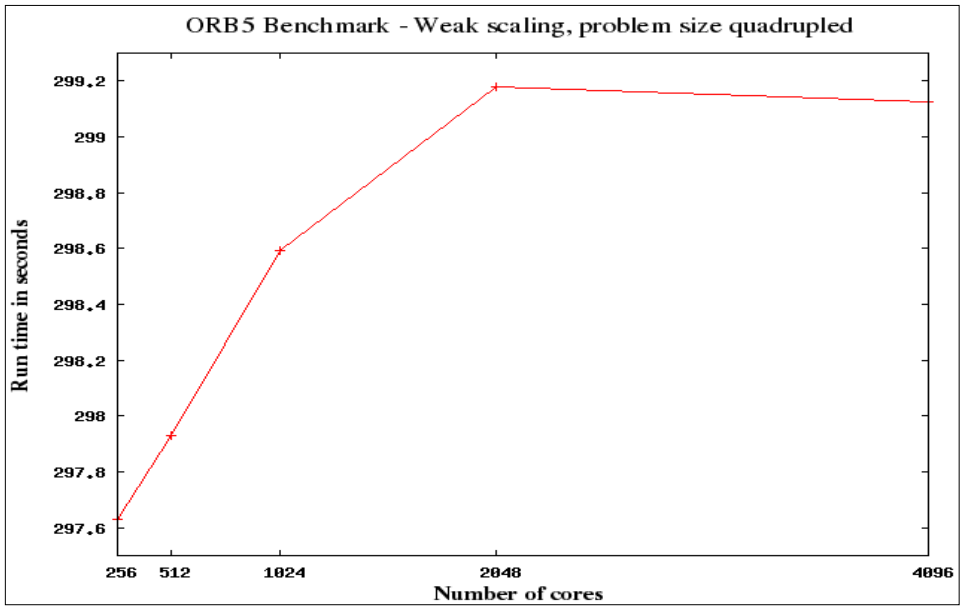


Fig. 5 Benchmark results for Weak scaling, Run time in seconds vs. Number of cores, problem size quadrupled

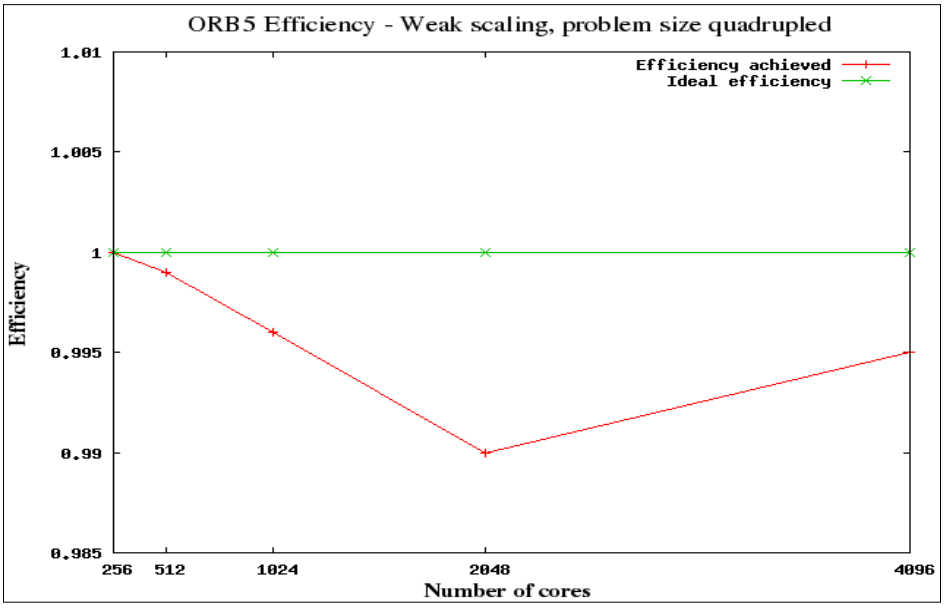


Fig. 6 Efficiency for Weak scaling, problem size quadrupled

2.2.2. GENE Code - Benchmark results on HPC-FF

The benchmark for the GENE code was done on HPC-FF up to a maximum of 4096 cores. The results of Strong scaling and Weak scaling have been given.

2.2.2.1. Strong scaling

The benchmark has been done on 512 cores up to 4096 cores. The speedup and efficiency have been calculated with respect to the timings obtained for 512 cores. Multiple runs have been done to ensure a good reliability of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 7, and the speedup in each case is given in Fig. 8.

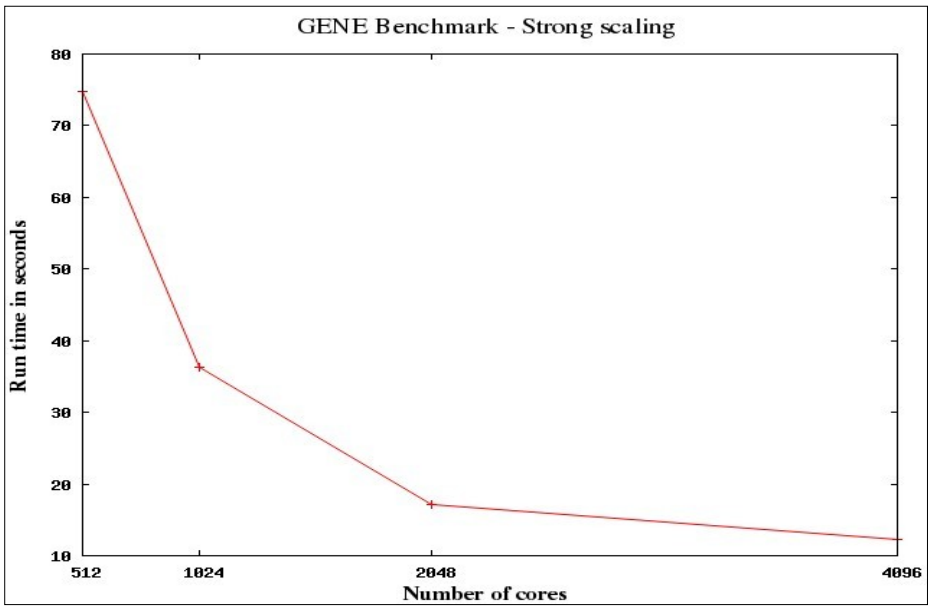


Fig. 7 Benchmark results for Strong scaling, Run time in seconds vs. Number of cores

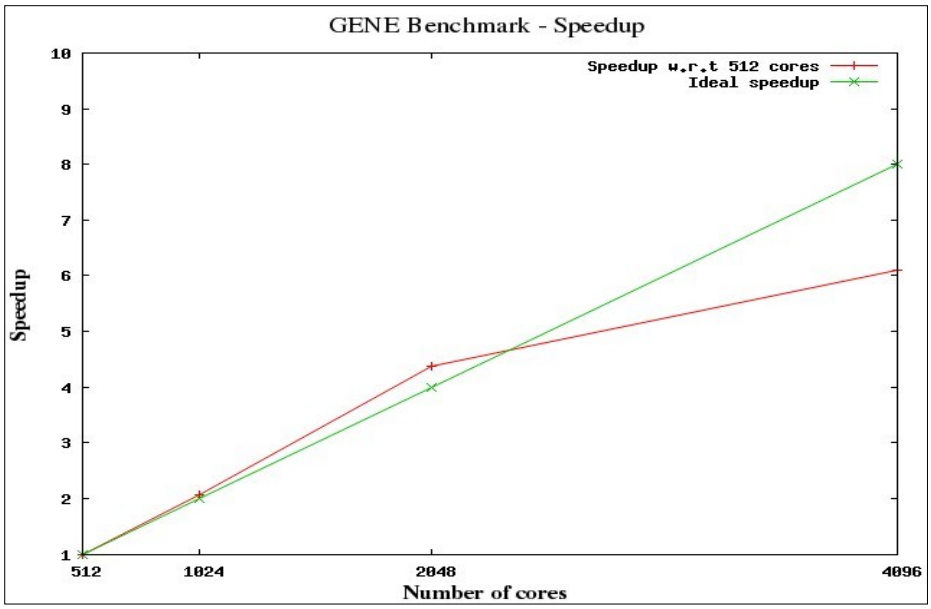


Fig. 8 Speedup for Strong scaling

2.2.2.2. Weak scaling

For weak scaling, the benchmark has been done on 512 cores up to 2048 cores. The problem size has been increased by a factor of 2 while doubling the number of processors. Runs for 4096 cores were not successful, which could be due to memory issues. The GENE code uses 1.752 GB of memory for weak scaling. Given that each core on HPC-FF has about 3GB, which is also used by MPI, the application could reach the limits of available memory for a large number of cores. The parallel efficiency has been calculated with respect to the timings obtained for 512 cores. Multiple runs have been done to ensure a good reliability of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 9 and the parallel efficiency is given in Fig. 10.

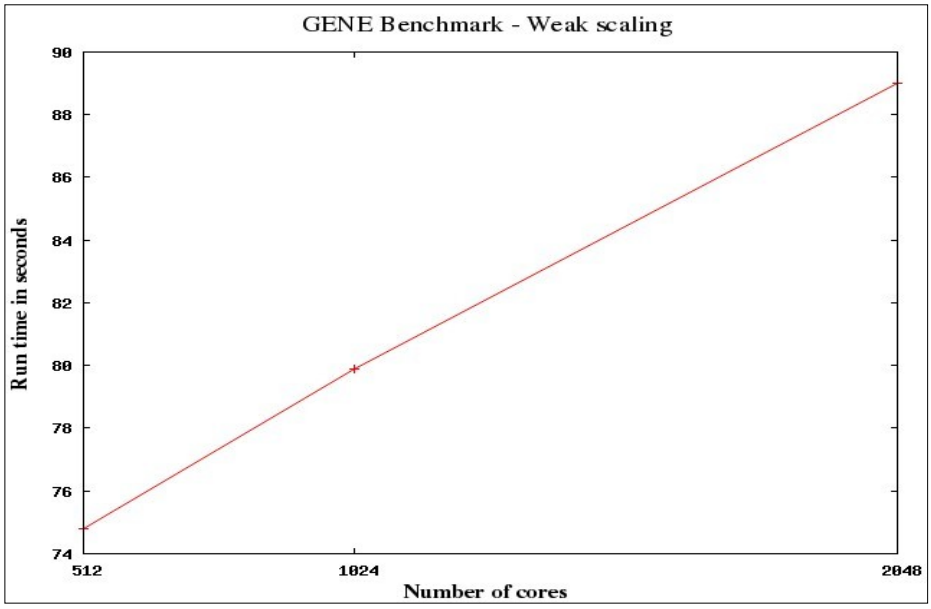


Fig. 9 Benchmark results for Weak scaling, Run time in seconds vs. Number of cores

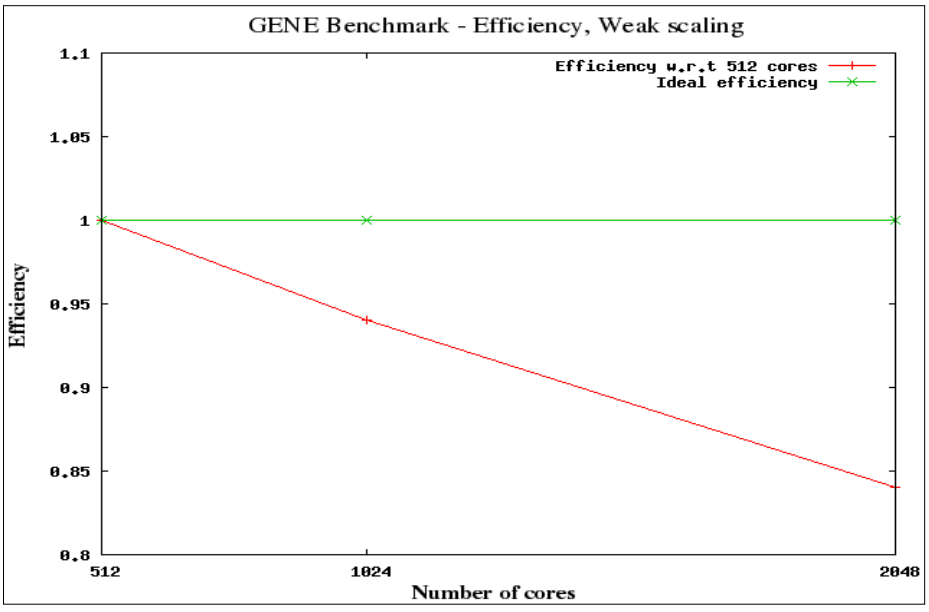


Fig. 10 Efficiency for Weak scaling

2.2.2.3. PSP_ONDEMAND and MPI_ANY_SOURCE

On HPC-FF, by default, MPI allocates all the required memory for its buffers before the start of the application. It is possible to change this behaviour using the environment variable PSP_ONDEMAND. Setting it to 1 allows MPI to allocate buffers dynamically during runtime. This also has the advantage that applications like GENE, which require a significant amount of the total memory, can be run on HPC-FF on a large number of cores, by setting PSP_ONDEMAND to 1. However, during some test runs for Strong scaling, it was found that the run times when PSP_ONDEMAND was set to 1 were considerably less to those obtained when PSP_ONDEMAND was set to 0. Since dynamic allocation of buffers, if done after the call to MPI_INIT, should ideally increase the run time, the results were not as expected. It was suggested by HPC-FF support that this could be due to the use of MPI_ANY_SOURCE in the MPI_Send calls. When PSP_ONDEMAND is set to 0, the communications library has to enquire a long list of receive buffers which results in an adverse impact on the run time. However, when PSP_ONDEMAND is set to 1, the library has to enquire a relatively lesser number of buffers which explains the better performance. The code was then changed to exclude the use of MPI_ANY_SOURCE.

The corresponding run times of the new code with PSP_ONDEMAND set to 0 and 1 were almost similar. Also, when the run times were compared to the old code which used MPI_ANY_SOURCE, it was found that:

- The run times for the new code when compared to the old code PSP_ONDEMAND set to 1 were similar.
- The run times for the new code when compared to the old code PSP_ONDEMAND set to 0 were much less.

These results were as expected. The timings are given in Fig. 11 and the speedup is given in Fig. 12. The figures include results only up to 2048 cores, runs on 4096 cores with PSP_ONDEMAND=0 were not successful due to memory issues as explained earlier.

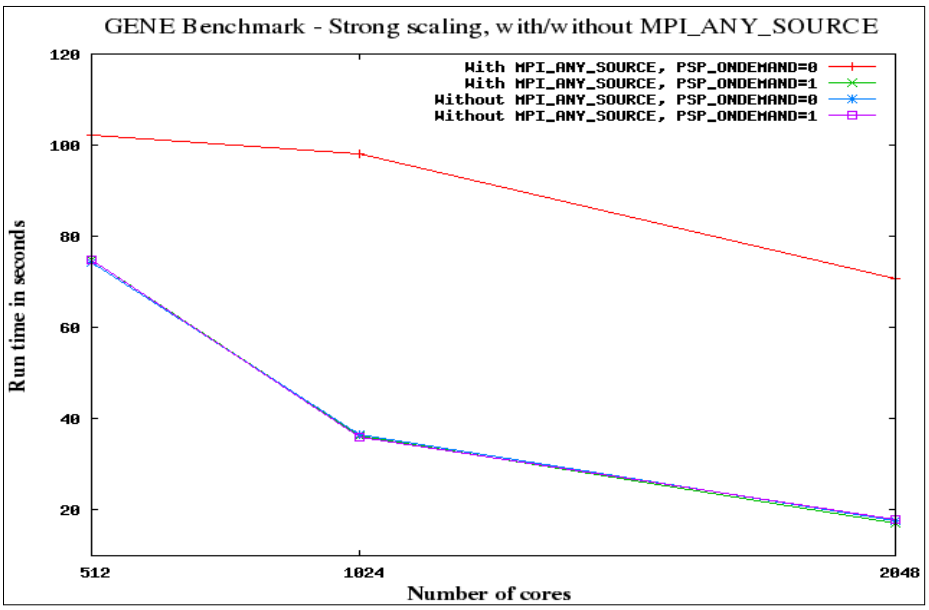


Fig. 11 Benchmark results for Strong scaling, timings with/without MPI_ANY_SOURCE. Run time in seconds, vs. number of cores

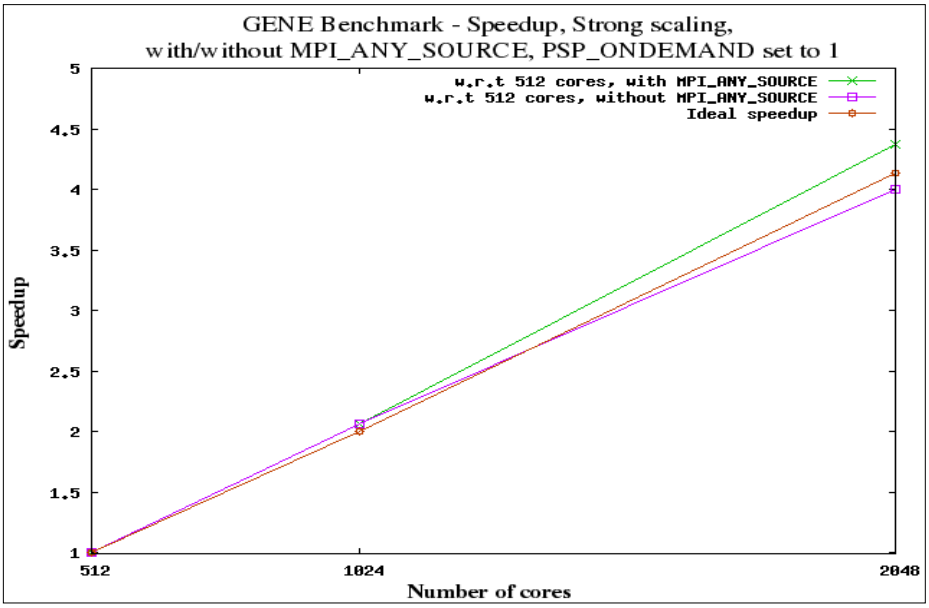


Fig. 12 Speedup, Strong scaling, with/without MPI_ANY_SOURCE, PSP_ONDEMAND set to one

2.2.3. GEMR Code - Benchmark results on HPC-FF

The benchmark for the GEMR code has been done on HPC-FF up to a maximum of 4096 cores. Three cases, AUG, JET and ITER have been tested. The results of Strong scaling and Weak scaling have been given.

2.2.3.1. Strong scaling

Strong scaling has been tested only for ITER. Two test cases, with varying grid sizes have been used. The benchmark has been done on 512 cores up to 4096 cores. The speedup and efficiency have been calculated with respect to the timings obtained for 512 cores. Multiple runs have been done to ensure a good representation of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 13, and the speedup in each case is given in Fig. 14.

On 4096 cores, with a grid size 2048x4096, a considerable amount of variation in run times was observed. Although, batch jobs are allocated complete nodes, the difference could be due to the order in which the I/O requests are serviced.

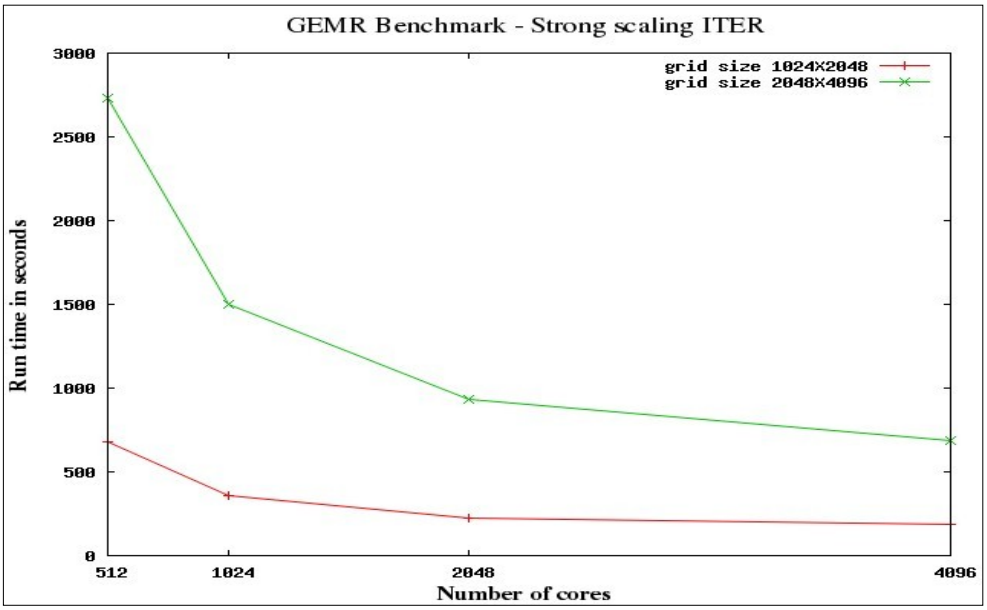


Fig. 13 Benchmark results for Strong scaling, Run time in seconds vs. Number of cores

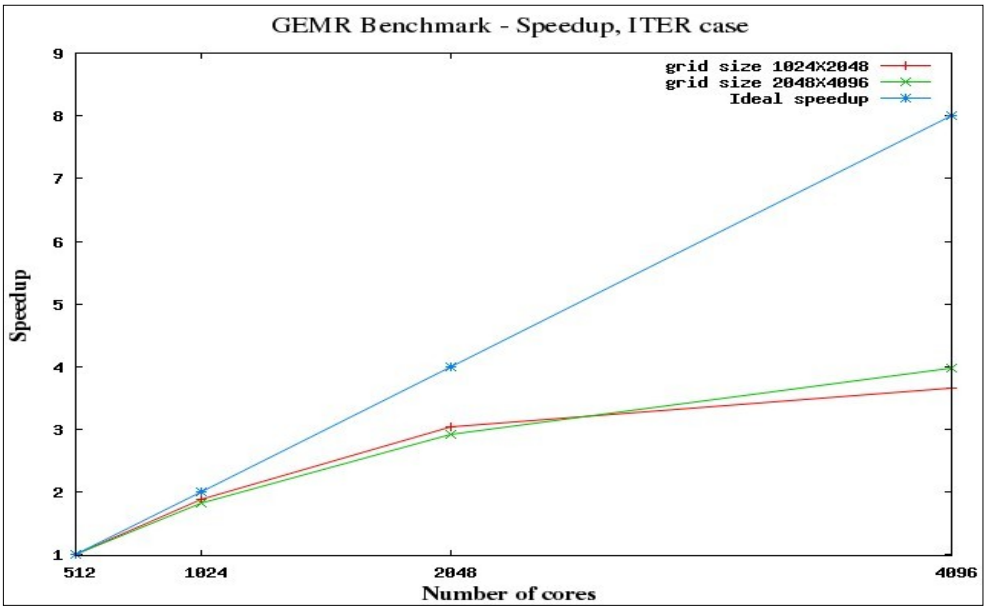


Fig. 14 Speedup for Strong scaling

2.2.3.2. Weak scaling

Weak scaling has been tested for the AUG, JET and ITER cases. Since the grid size increases by a factor of 4 for each case, the number of processors was increased by a factor of 4 for each case. AUG was run on 256 cores, JET on 1024 and ITER on 4096 cores. The parallel efficiency has been calculated with respect to the timings obtained for the AUG case, i.e., 256 cores. Multiple runs have been done to ensure a good representation of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 15 and the parallel efficiency is given in Fig. 16.

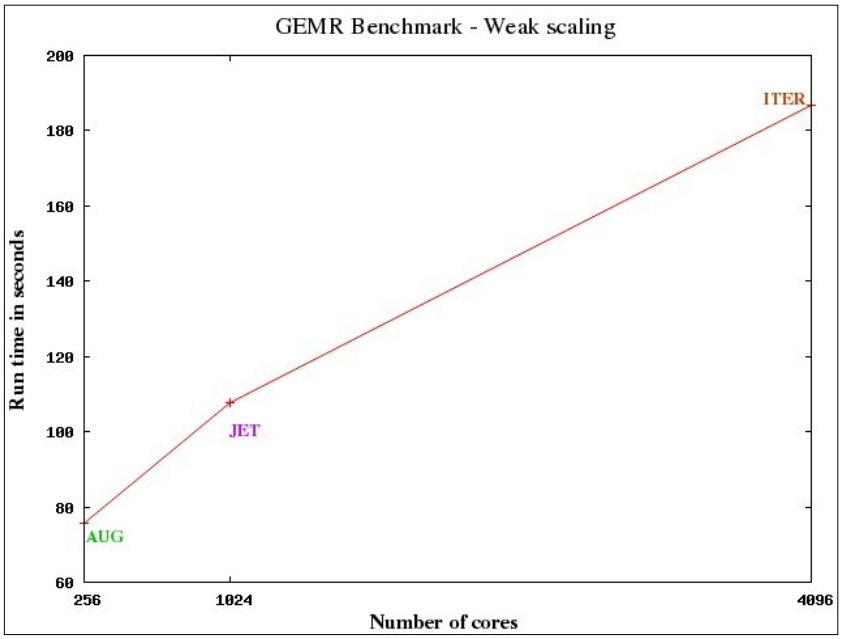


Fig. 15 Benchmark results for Weak scaling for AUG, JET and ITER cases, Run time in seconds vs. Number of cores

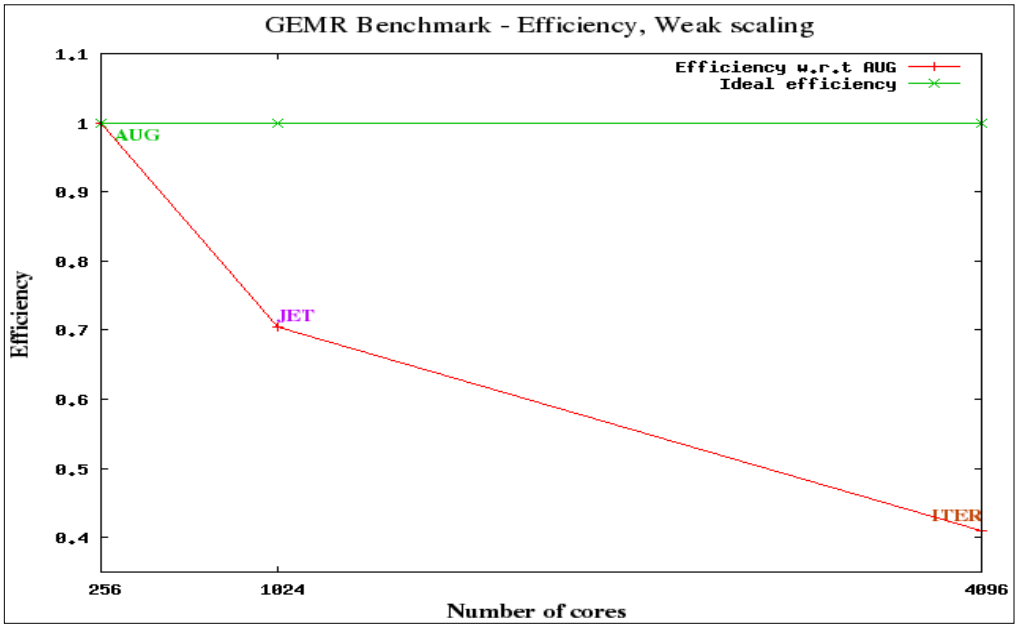


Fig. 16 Efficiency for Weak scaling, AUG, JET and ITER cases

2.2.4. JOREK Code - Benchmark on HPC-FF

The JOREK code was ported onto HPC-FF to be benchmarked as part of the BEUPACK benchmark suite. JOREK is a hybrid code that uses both MPI and OpenMP directives. It uses a number of external libraries including the PaStiX library which is a parallel solver for large sparse systems. HPC-FF uses the ParTec MPI implementation by default.

JOREK has a call to MPI_Thread_Init which requests a certain level of MPI Thread level support. Depending on the configuration of the MPI library, it provides a suitable level that is either the same or lower than the level requested. We noticed that, JOREK requests for the MPI_THREAD_MULTIPLE (MTM) level by default and this could be changed by the user. With MTM, multiple threads can call MPI with no restrictions. However, it was unclear initially as to why the highest level of thread safety for MPI was requested by JOREK.

The code ran through on HPC-FF for 1 MPI and 1 OpenMP thread. While running the code in the hybrid mode, for example, on 12 nodes with 1 MPI process and 8 OMP threads per node, it terminated with a deadlock. This behavior was found to be random; there were instances when the code ran through successfully without any deadlocks. There were no issues when JOREK was run as a pure MPI application. Using multiple OMP threads in combination with many MPI tasks seemed to raise the deadlock issue.

We then found that the PaStiX library can be compiled in two different versions to use different levels of MPI Thread support, the MTM or MPI_THREAD_FUNNELED (MTF). With MTF, only the main thread can make a call to the MPI library. In PaStiX as well, the default is the MTM level.

Among the libraries that JOREK uses, only PaStiX requires a specific MPI Thread level support. Hence, to find the reason for the deadlock, we had to concentrate on JOREK and PaStiX. There were some tests carried out to find the cause for the problem. Sections 2.2.4.1 to 2.2.4.3 detail the findings and the measures taken to get JOREK running on HPC-FF.

2.2.4.1. PaStiX

To exclude any problems caused due to the PaStiX library, we tried to run the example programs provided by the PaStiX library. These examples did not provide the same scenario as JOREK since some of the programs either used only MPI, or allowed only one MPI process with multiple threads. However, the tests were done to rule out any fundamental issues with the library itself.

ParTec MPI only provides support up to MPI_THREAD_SERIALIZED (MTS), where multiple threads can call MPI but only one at a time. MTS has a support level in between MTF and MTM. Since MTM was not available with ParTec MPI and PaStiX could be compiled with MTF, we tested PaStiX with the MTF level. It was found that, on 1 MPI process, the example programs ran through successfully. But running the examples with multiple MPI processes caused the execution to fail even with 1 OMP thread. The programs terminated with an error related to persistent communications and MPI_Start/MPI_Startall.

2.2.4.2. MPI library

To ensure that the problems faced were not due to JOREK or PaStiX, JOREK was ported to two other systems at the Rechenzentrum Garching (RZG) - the IBM POWER6 machine and to the SUN Linux cluster “AIMS”.

The IBM POWER6 machine has the Parallel Operating Environment (POE) with its inbuilt MPI library, with both the MTF and MTM options that the PaStiX library and JOREK use. We were able to run the code on this machine on multiple MPI processes with multiple OMP threads. Different test cases, with either the MTF or MTM option in both the PaStiX library and in the JOREK code, were run. The tests in each case were successful.

The code was then ported to the SUN Linux cluster “AIMS”, which has Intel MPI and a hard- and software configuration similar to HPC-FF. Intel MPI also has both the MTF and MTM options available. Here, as well, the code ran through with all possible combinations of MPI and OMP threads with both MTF and MTM levels.

It was found that the major difference between the IBM POWER6 machine, the SUN Linux cluster “AIMS” and HPC-FF was that the former two machines could provide the MTM thread level support. Since ParTec MPI only provides support up to MTS, this meant that using a thread level less than MTM for either JOREK or the PaStiX library could be a reason for the deadlock. Hence, on request, Intel MPI was installed on HPC-FF.

The JOREK code, when run on Intel MPI with MTM level enabled on JOREK and PaStiX, was able to run successfully on HPC-FF with multiple MPI processes and multiple OMP threads each. It also ran through with the MTF level on Intel MPI with MTF level enabled on JOREK and PaStiX

To verify that the MTM level was the only cause of the problem, another version of ParTec MPI was installed with MTM support enabled. However, this was also found to cause a deadlock in JOREK.

2.2.4.3. Conclusions for JOREK on HPC-FF

To summarize, we were able to run JOREK on Intel MPI with either MTF or the MTM level set in both JOREK and PaStiX. JOREK, however, did not run through on ParTec MPI.

Hence, we have two open calls with ParTec. One considers the provision of a MPI which provides a functioning MTM version. And the other one to find out why the MTF version of the PaStiX library results in a MPI_Start/MPI_Startall error. As long as these issues are not solved JOREK can be only run in the hybrid MPI/OpenMP mode with Intel MPI on HPC-FF.

2.2.5. MDCASK Code - Benchmark results on HPC-FF

The benchmark for the MDCASK code has been done on HPC-FF up to a maximum of 4096 cores. The results of Strong and Weak scaling have been given.

2.2.5.1. Strong scaling

The benchmark has been done on 512 cores up to 4096 cores. Problem sizes of $250 \cdot 10^6$, $500 \cdot 10^6$ and $1 \cdot 10^9$ atoms have been tested. For 1 billion atoms, the problem size is too large to fit into the memory of 512 cores, hence benchmarks for this case have been done on 1024 cores or more. The speedup and efficiency have been calculated with respect to the timings obtained for 512 cores. Multiple runs have been done to ensure a good representation of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 17, and the speedup in each case is given in Fig. 18.

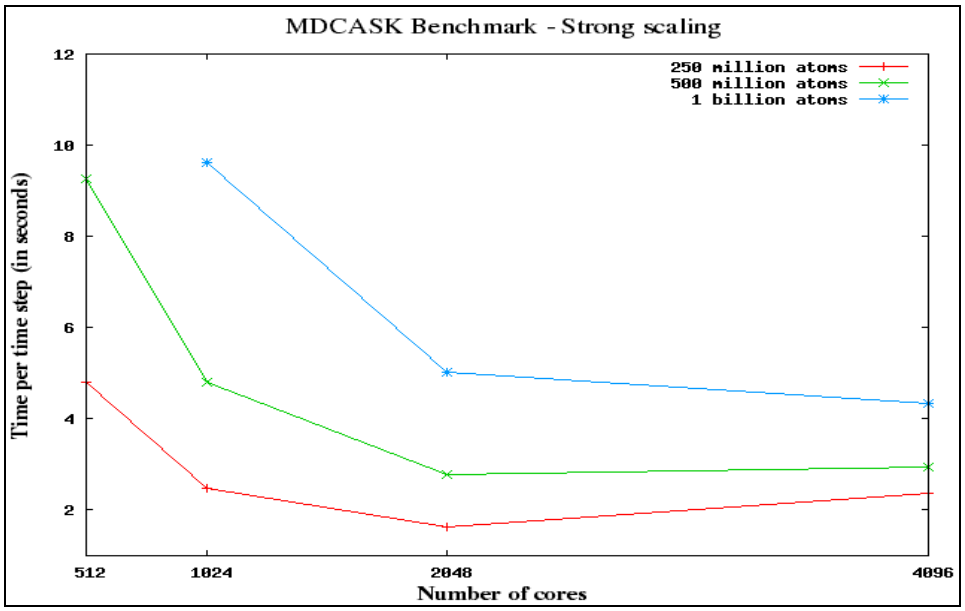


Fig. 17 Benchmark results for Strong scaling, Time per time step (in seconds) vs. Number of cores

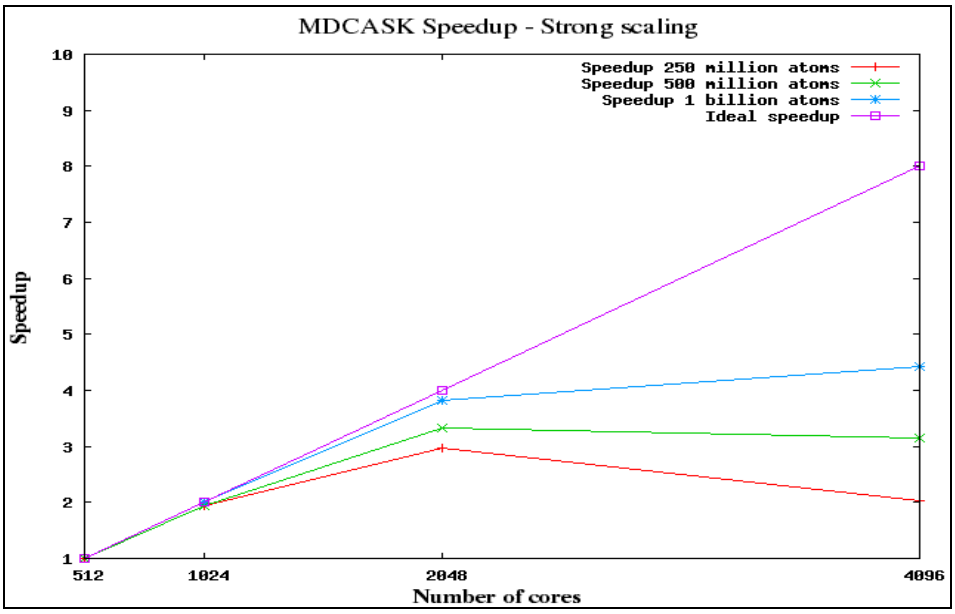


Fig. 18 Speedup for Strong scaling

2.2.5.2. Weak scaling

The benchmark has been done on 512 cores up to 4096 cores. Problem sizes of $250 \cdot 10^6$, $500 \cdot 10^6$, $1 \cdot 10^9$ and $2 \cdot 10^9$ atoms have been tested. The speedup and efficiency have been calculated with respect to the timings obtained for 512 cores. Multiple runs have been done to ensure a good representation of results. The median of the timings obtained have been plotted. The benchmark results are given in Fig. 19 and the efficiency in each case is given in Fig. 20.

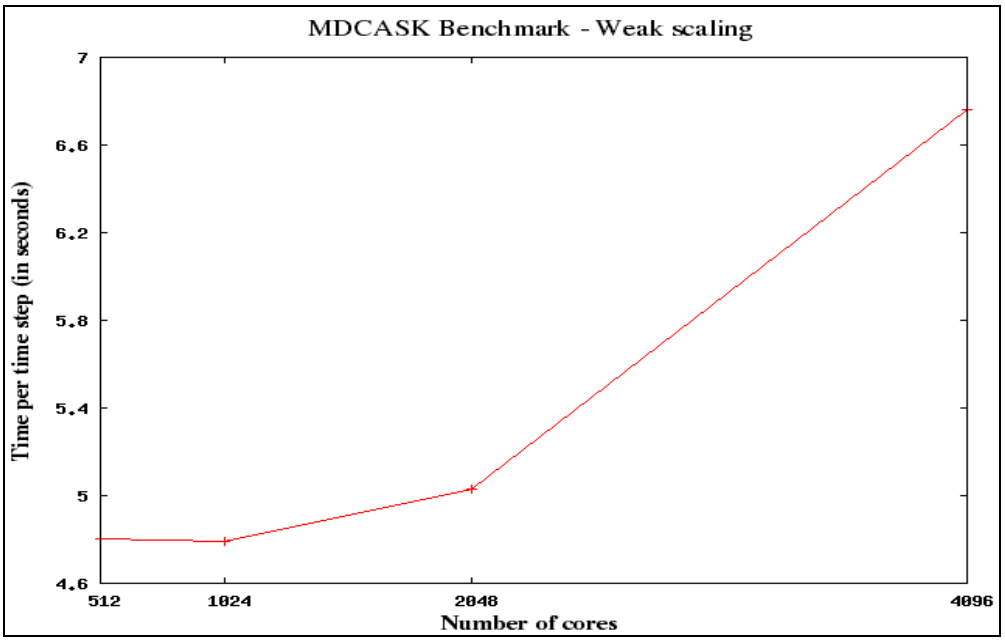


Fig. 19 Benchmark results for Weak scaling, Time per time step (in seconds) vs. Number of cores. Problem size starts with $250 \cdot 10^6$ atoms for 512 cores and doubles thereon

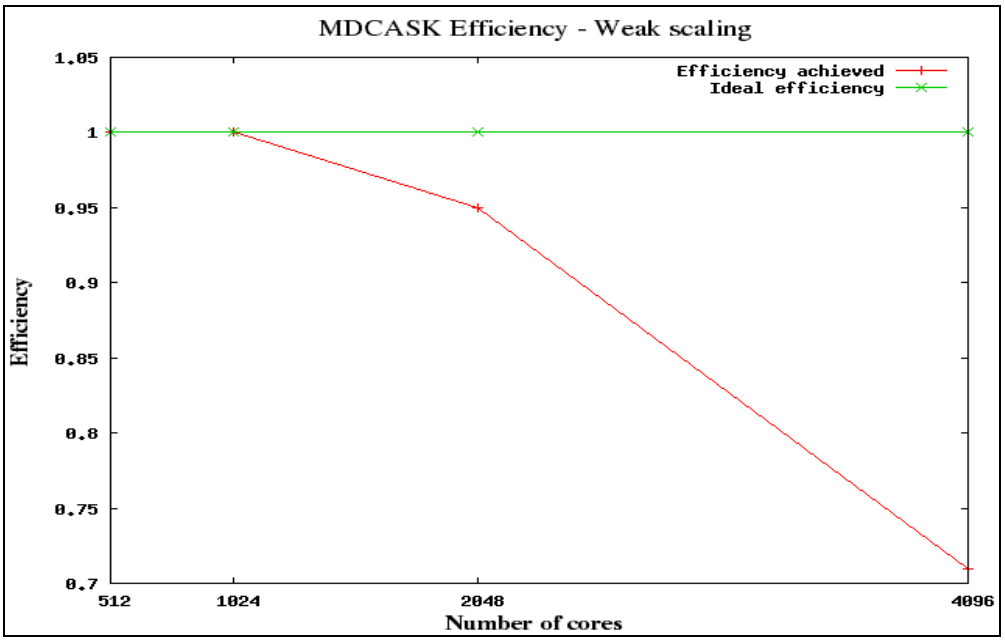


Fig. 20 Efficiency for Weak scaling. Problem size starts with $250 \cdot 10^6$ atoms for 512 cores and doubles thereon

2.2.5.3. Notes on MDCASK Code

- On HPC-FF, the code timed out when run on more than 512 cores, using ParTec MPI. It was found that running the code with PSP_ONDEMAND set to 1, which allocates MPI buffers dynamically during run time, could cause this problem if a code uses all-to-all communications. Instead a test case with PSP_ONDEMAND set to 0 on 1024 cores for $500 \cdot 10^6$ atoms was successful. However, it is also known that the amount of memory occupied by ParTec MPI when PSP_ONDEMAND is set to 0 is large enough to cause memory issues for large problem sizes. Hence, Intel MPI was used for all test cases.

- The code required a change in the way the calculation of Average CPU time per time step per atom was done. In the original code, it was done by dividing the Cumulative CPU time with the product of total number of time steps and total number of atoms. For large numbers such as $500 \cdot 10^6$, the product of 10 (the number of time steps) and $500 \cdot 10^6$ (number of atoms) is larger than the maximum value a 32 bit integer can hold, i.e., $2^{31}-1 = 2147483647$. This caused a floating point overflow and resulted in negative values being output for the Average CPU time. The code below

  *Average CPU time per time step per atom = Cumulative CPU / (no. of time steps * total no. of atoms)*

  was changed as follows:

  Average CPU time per time step per atom = Cumulative CPU / (no. of time steps)/ (total no. of atoms)

  with the number of time steps and atoms being cast to real. This gave the expected results.

- The code could be run on a maximum of 2 billion atoms. Exceeding this number also exceeds the maximum value a 32 bit integer can hold, i.e., $2^{31}-1 = 2147483647$. This in turn causes an overflow and results in error during execution.

2.2.6. GYSELA Code - Benchmark results on HPC-FF

The benchmark for the GYSELA code has been done on HPC-FF up to a maximum of 4096 cores. The results of Strong and Weak scaling have been given.

2.2.6.1. Strong scaling

The benchmark has been done on 512 cores up to 4096 cores. The speedup and efficiency have been calculated with respect to the timings obtained for 512 cores. The benchmark results are given in Fig. 21 and the speedup in each case is given in Fig. 22.

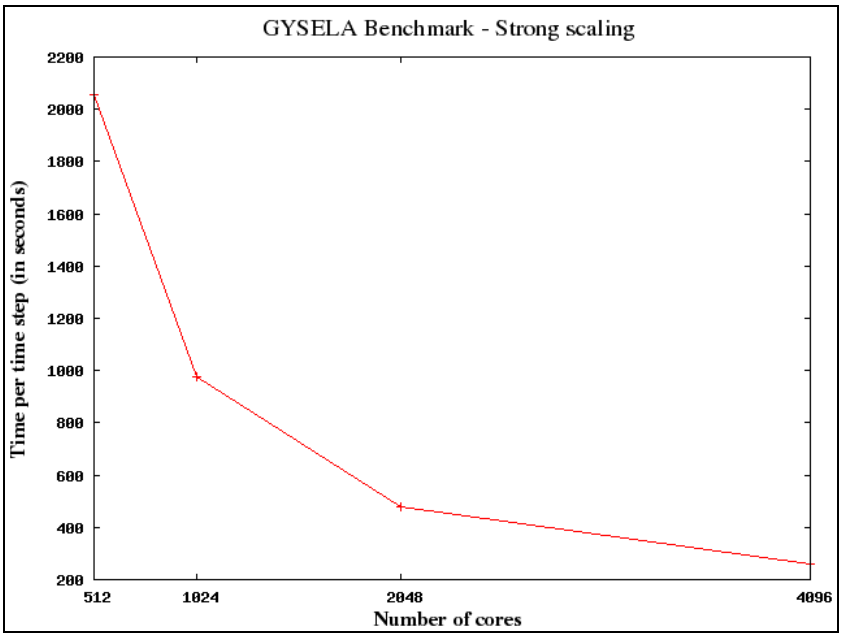


Fig. 21 Benchmark results for Strong scaling, Time per time step (in seconds) vs. Number of cores

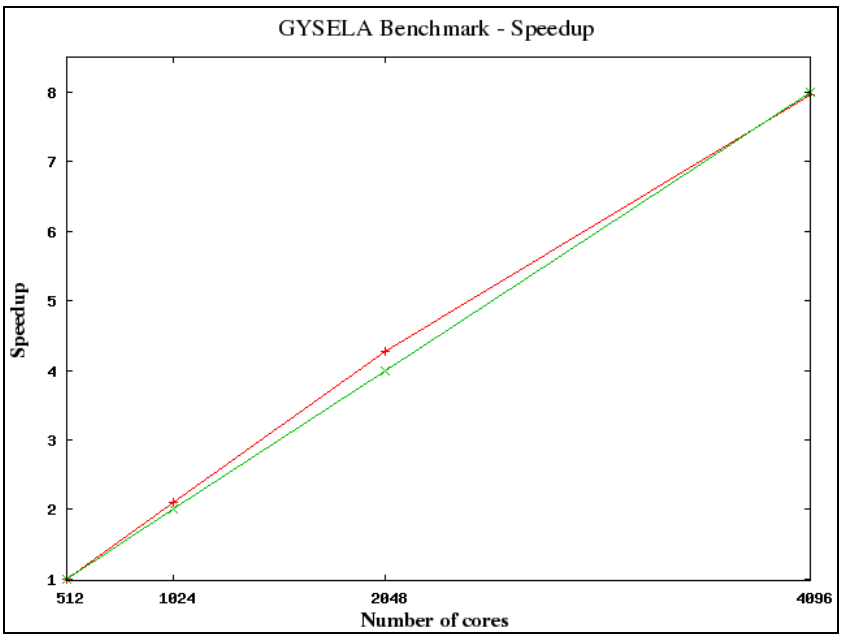


Fig. 22 Speedup for Strong scaling (red curve). Linear scaling is given by the green curve

2.2.6.2. Weak scaling

The benchmark has been done on 512 cores up to 4096 cores. The benchmark results are given in Fig. 23 and the efficiency in each case is given in Fig. 24.

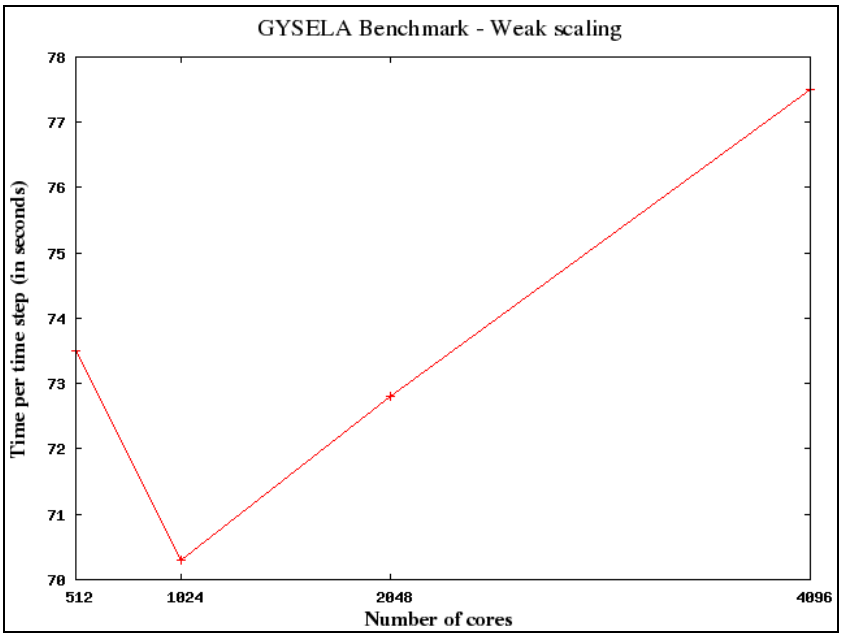


Fig. 23 Benchmark results for Weak scaling, Time per time step (in seconds) vs. Number of cores

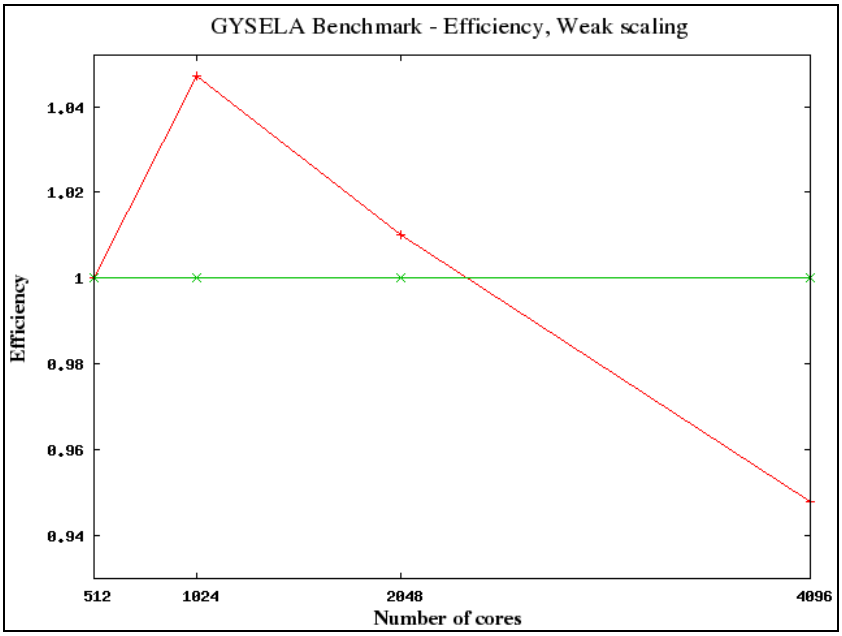


Fig. 24 Efficiency for Weak scaling (red curve). Constant efficiency is given by the green curve

3. Report on GYGLES project

3.1. Introduction

The GYGLES code solves the linear gyrokinetic equations for electrons and one or more ion species in full tokamak geometry. The original code version was developed at CRPP. At that stage it was restricted to an electrostatic model with adiabatic electrons (Fivaz et al., 1997). Further code development took place at IPP, e.g. the extension to an electromagnetic model (Mishchenko, Hatzky, Könies, 2004) and the implementation of a generalized quasi-neutrality equation solver. Before the polarization density term in the quasi-neutrality equation was simplified by the assumption of the so-called long wavelength approximation. Now the generalized solver solves the integral quasi-neutrality equation accurate to all orders of the perpendicular wave number (Mishchenko, Könies, Hatzky, 2005). The precise description of the polarization density and current should be important for the boundary-layer dynamics of the tearing and kink modes which develop extremely fine radial structures at the position of singularities (on the rational flux surfaces).

However, the code had its limitations when it came to larger domain sizes in combination with the generalized solver mentioned above. In such a case the number of non-zero elements in the quasi-neutrality matrix increases significantly. Hence, the character of the matrix changes from a more dense to a more sparse type. As a consequence, the implemented algorithms for solving dense matrix equations, i.e. LAPACK and ScaLAPACK were no longer appropriate for such cases. Instead, a direct sparse solver came into the focus of investigation as e.g. the IBM Watson Sparse Matrix Package (WSMP).

3.2. Cleaning, porting and testing the code

The code is written in Fortran 95 and consists of approximately 40,000 code lines localized in about 30 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 analyses the set of source files and constructs automatically a consistent dependency list.

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

The code package consists of the code GYGLES and its data analyzing tool ANGY. Both components have been ported to HPC-FF to give the possibility of checking the complex data output with the ANGY tool. As a consequence the graphic package XGrafix needed by ANGY had to be installed in addition.

Of course, it is necessary to test large codes such as GYGLES on several different computing architectures to reveal programming bugs at run time. The capability of the different compilers for runtime error detection via compiler options is strongly depending on the software vendor. Accordingly, the GYGLES code has been tested on many different platforms with a variety of Fortran 95 compilers:

- Linux clusters with Intel 64 (Extended Memory 64 Technology) and the compilers from Intel and Lahey/Fujitsu
- IBM POWER6 575 system and the IBM XLF compiler

- IBM BlueGene/P and the IBM XLF compiler

Several programming bugs introduced during the development process have been found and corrected.

3.3. The implementation of the IBM Watson Sparse Matrix Package

The IBM Watson Sparse Matrix Package, WSMP, (http://www-users.cs.umn.edu/~agupta/wsmp.html) is a high-performance software package for solving large sparse systems of linear equations using a direct method. It has the advantage that it can be used as a serial package, or in a shared-memory multiprocessor (SMP) environment, and/or as a scalable parallel solver in a message-passing (MPI) environment. It shows very good scalability, is available for AIX and Linux operating systems and is continuously improved.

We initiated the compilation of a WSMP version for the Lahey/Fujitsu compiler to perform the runtime test during development mentioned in Sec. 3.2. In addition, it became necessary to recompile the WSMP library for the Nehalem processor on HPC-FF which was done by the IBM developer of WSMP, Anshul Gupta, himself. A close collaboration with Anshul Gupta became mandatory to amend many WSMP implementation problems which occurred during the project. The reasons were twofold. On the one hand, typical WSMP users have matrices with real arguments so that its counterparts with complex arguments used in GYGLES were not as well tested. And on the other hand, we defined our own MPI communicator which is also not the standard approach of typical users. As a consequence around 100 e-mails have been exchanged over the project time.

The Hermitian matrices in the original GYGLES version are built in the initial phase and stored as band matrices. This format is then converted in two steps. First it is converted into the general sparse format which stores for every non-zero element its value and column and row indices. Afterwards the lower triangular part of the matrix is converted into the compressed sparse column (CSC-LT) format used by WSMP. Controlled by an option in the input file the whole matrix information can be written to a file. Hence, a testing of the WSMP solver became possible in a modular manner. The only information needed to be passed was the matrix information stored in the matrix file. Then an independent testbed was used to evaluate operability, scalability, maximum number of usable processors and memory consumption of the WSMP solver.

3.3.1. The implementation of the WSMP testbed

The testbed includes a serial and a parallel MPI version of WSMP with two solvers using a Cholesky and LDL* decomposition. The latter has the advantage that it seems to be more robust. This can be advantageous for the decomposition of the quasi-neutrality matrix as this matrix is built up by a Monte Carlo method and suffers from statistical noise.

The implementation of the parallel MPI version of WSMP was done in the following way. The set of processors was divided into subgroups of processors by defining a corresponding MPI communicator. Each subgroup makes its own parallel solve of the matrix equation. This is necessary because usually the total number of processors is much larger than the maximal usable number of processors for a parallel WSMP solve. Hence it is more efficient to solve the matrix equation simultaneously in several subgroups of processors as to just use one subgroup and then to broadcast the solution to all the remaining processors. Both a 0-master and a peer mode have been investigated. In the 0-master mode the right-hand-side and the matrix are passed exclusively by processors 0 to the parallel WSMP solver routine. In return the

solution is also passed to the processor 0 and has to be broadcasted afterwards to the other processors in the subgroup. More efficient is the peer mode where all processors pass a part of the right-hand-side and of the matrix and get in return a part of the solution vector. In this case only an all-gather MPI call is necessary to finally broadcast the solution vector to every processor in the subgroup.

In addition, the testbed was used to test the serial and parallel sparse matrix vector multiplication included in WSMP. Such an operation is needed in GYGLES when a special noise suppression algorithm (control variate) is used.

After successful evaluation and debugging of WSMP for many matrices of interest the testbed was rewritten as a module to be included in the GYGLES program.

3.4. Changes in the GYGLES code

The GYGLES program structure was reorganized. It consists now of separate modules for the matrix building and the solvers, i.e. LAPACK, ScaLAPACK and WSMP. In addition, the matrix data structure has been enhanced from a Fortran 77 to a Fortran 95 style. Instead of allocating a large array for the maximum possible number of matrices the allocation process can be handled much more flexibly now. Only memory is allocated for matrices which are actually needed for a specific simulation regime. And for each matrix only the necessary amount of memory is allocated and not the memory needed for the largest matrix. This can save a lot of memory as the individual sizes of the matrices differ by orders of magnitude. And finally band matrix memory is deallocated after usage if no longer required.

3.5. Performance of the WSMP solver

The parallel solver from WSMP has been benchmarked against the ScaLAPACK solver. The test case was a typical electrostatic simulation using the generalized quasi-neutrality solver. The grid size was n_s=128 by n_θ=64 and 16,000,000 ion markers have been used.
First it revealed that the memory consumption of the WSMP version is 10 times smaller than the ScaLAPACK version (134 MB vs. 1386 MB). This is due to the fact that the ScaLAPACK solver can make no use of the sparse structure of the matrix. In addition, the memory consumption of the original GYGLES code would have been larger by a factor of two due to the over-allocation of memory for the matrices discussed in Sec 3.4. As a result the test case would not have fitted into the memory of the HPC-FF compute nodes which have less than 3 GB per core. Thus, the memory reduction is remarkable.

Second the WSMP solver can use up to eight processors in parallel while the ScaLAPACK solver is limited to just two processors (see Fig. 25). This makes the WSMP solver approximately six times faster than the corresponding ScaLAPACK solver.

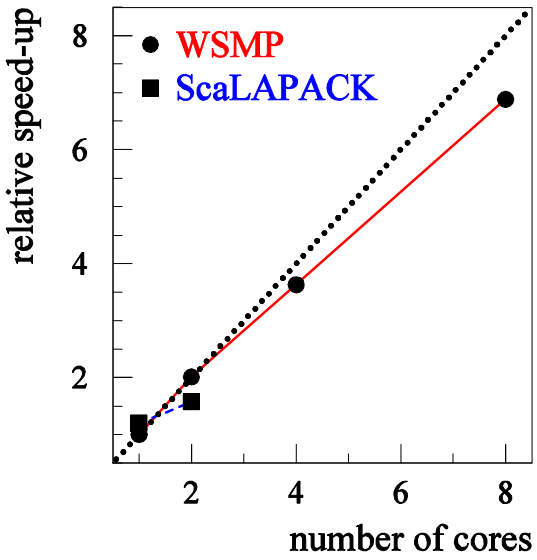


Fig. 25 Strong scaling of the parallel WSMP solver for a generalized quasi-neutrality equation case with a grid of 64 x 128 quadratic B-splines on the HPC-FF computer with 2 Intel Nehalem-EP 2.93 GHz quad-core processors on each node located at JSC. red curve: relative speed-up of the WSMP solver; blue curve: relative speed-up of the ScaLAPACK solver; dotted curve: optimal speed-up

For a strong scaling up to large numbers of processors the markers are distributed on more and more processors. Hence, the work load of the markers becomes so small that the matrix solving process becomes the dominating part in the single core execution. Hence, a poorly scaling solver part will finally limit the overall scaling due to Amdahl's law. This can be clearly seen in Fig. 26. The GYGLES code using the ScaLAPACK solver only scales up to 128 cores (parallel efficiency of 80 %). Instead the WSMP solver version scales up to 512 cores (parallel efficiency of 79 %). Thus, the WSMP solver version can efficiently use four times as many cores on the same problem size.

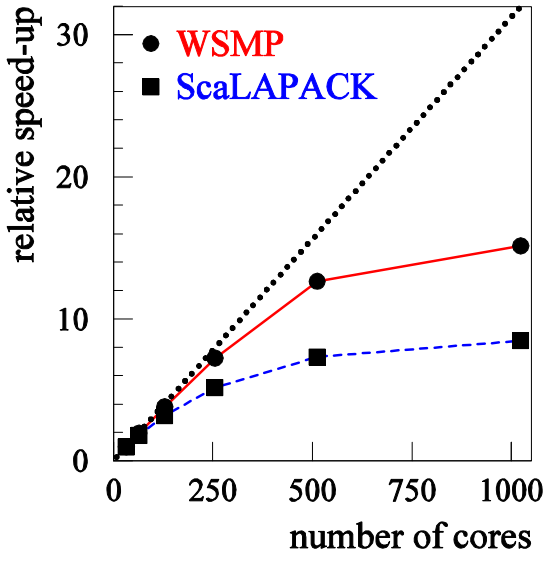


Fig. 26 Strong scaling of the GYGLES code for a generalized quasi-neutrality equation case with a grid of 64 x 128 quadratic B-splines and 16,000,000 ion markers on the HPC-FF computer with 2 Intel Nehalem-EP 2.93 GHz quad-core processors on each node located at JSC. red curve: relative speed-up of the GYGLES code with the WSMP solver; blue curve: relative speed-up of the GYGLES code with the ScaLAPACK solver; dotted curve: optimal speed-up

3.6. Conclusions for GYGLES project

The GYGLES proposal had requested support for the optimization of the matrix equation solver part. This is of interest for example when small scale structures in the range of the ion gyro-radius or below should be resolved. Such simulations have to take an exact representation of the polarization density term in the quasi-neutrality equation into account. As a consequence the formerly implemented band matrix solvers from LAPACK and ScaLAPACK became inappropriate. Instead, the sparse solver from WSMP has been implemented and the memory allocation for the matrices has been optimized. For a typical test case the memory consumption for the matrix equation solving could be reduced by a factor of 20 and be sped up by a factor of six. Without such a reduction in memory consumption the test case would have not fit into the 3 GB memory per core of HPC-FF. Finally, this results in a much better scalability of the GYGLES code by a factor of four to a total number of 512 cores.
In future the implementation of an adaptive mesh into GYGLES should be taken into account. Relevant studies have been done by the ORBIS project and resulted in an extension of the BSPLINES Fortran 95 module.

4. Report on OPTGS2 project

4.1. Introduction

The long-term objective of the GS2 (http://gs2.sourceforge.net/) 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.

However, it was not advisable to start such a demanding task dealing with the full dimensionality (five dimensions) and complexity of the GS2 code. Instead a (1+1) dimensional model problem was specified, which reflects the numerical problem being present in GS2. It is essentially an advection problem with a distribution function in the coordinate's space *s* and velocity *v* plus an integral source term over velocity space. For the beginning an even simpler test problem has been selected - the well known one-dimensional linear advection equation. It has the advantage that analytic solutions are available for arbitrary initial conditions.

The basic idea of the Galerkin finite element method is to discretize the differential equation by so-called finite elements which got their name from their finite support. In contrast to the continuous Galerkin finite element method (CG-FEM) the finite elements of the DG-FEM method are discontinuous and not continuous at their boundaries. As a consequence numerical fluxes evolve over the finite element boundaries which lead to additional terms in the numerical discretization of the scheme. Therefore, DG-FEM methods have a hybrid character which places them in between finite volume schemes on the one hand and finite element schemes on the other hand. The advantages are that DG-FEM methods can be applied to complicated geometries and boundary conditions while they are closely linked to the control volume approach of finite volume methods.

As a matter of Godunov's theorem only the linear first order "upwind" scheme has a high enough intrinsic viscosity to preserve monotonicity which prevents the evolution of spurious oscillations (wiggles) at steep gradients of the solution. All other linear high order schemes suffer from these numerical artefacts. To overcome these drawbacks, various high-resolution, non-linear techniques have been developed, often using flux/slope limiters. These numerical methods usually have the property of being total variation diminishing (TVD). A TVD scheme is monotonicity preserving, if the following properties are maintained as the solution is evolved in time:

- No new local extrema can be created within the solution spatial domain.
- The value of a local minimum is non-decreasing, and the value of a local maximum is non-increasing.

In addition to TVD there are more relaxed constraints as e.g. the total variation diminishing in the mean (TVDM) which restricts TVD only to the cell averaged values and total variation bound (TVB) which prevents the total growth of wiggles by setting a supremum.

Standard DG-FEM schemes are not intrinsically TVD schemes. Hence, at steep gradients of the solution wiggles do occur. In order to make the scheme TVD special demands have to be made on the time integration scheme and the fluxes across cell surfaces. For further details please see the technical report "*Combining Runge-Kutta*

discontinuous Galerkin methods with various limiting methods" which is accessible under the URL: http://www.efda-hlst.eu/tutorials.

4.2. Finite difference schemes with flux limiter

To provide a testbed for TVD schemes the 1D advection equation was first discretized by finite differences. A spatial "upwind"/centered discretization was implemented in combination with different time integration schemes, e.g. an explicit scheme, a Crank-Nicholson scheme, a fully implicit (Eulerian backward) scheme and higher order TVD-Runge-Kutta explicit time discretization schemes.

Numerical tests confirmed the aforementioned fact that except for the first order "upwind" scheme all higher order methods have the tendency to evolve spurious oscillations when steep gradients of the solution occur. Different flux limiting methods were investigated which assemble the numerical flux as a blend of a lower order and a higher order numerical flux. A higher order flux is used by default but in regions where steep gradients occur, the method turns over continuously to a lower order flux, which exhibits a larger amount of numerical viscosity. As a result wiggles are damped away and TVD is assured.

4.3. Continuous Galerkin schemes

In addition, the continuous Galerkin and the Petrov-Galerkin finite element method, respectively, have been used to discretize the 1D advection equation. Using B-splines of different order as finite elements a total number of six schemes have been derived. Both Dirichlet and periodic boundary conditions have been implemented and tested with weighted averages of the explicit and implicit Euler's scheme. In case of a periodic boundary condition a stability analysis in time was performed. Each time integration scheme has a typical region of absolute stability in the complex plane. As long as all eigenvalues of the matrix which represents the numerical scheme are within the stability region the scheme is proven to be stable.

The B-spline CG-FEM scheme is formally equivalent to a compact finite difference scheme of very high order. Due to the high approximation order, spurious oscillations also occur with this scheme. It is not obvious how to control these wiggles for such a scheme. Thus, it became clear that these schemes were not suited for our purpose although they have a very high approximation order. Instead, we put our focus on the DG-FEM schemes as these schemes have a high approximation order with established methods for wiggle control.

4.4. Discontinuous Galerkin schemes for the 1-dim model problem

A typical DG-FEM scheme discretizes the one dimensional advection equation by using Legendre polynomials up to a maximum order p as finite elements. In principle, our implementation is not restricted to a certain maximal p. However, in practice we restricted ourselves to $p=2$, i.e. quadratic polynomials and could achieve a very good numerical approximation to the exact solution especially if a square wave was used as initial condition. Of course, such a high approximation order produced wiggles in the numerical solution again. So we had to apply several algorithmic improvements to get this problem under control.

4.4.1. Explicit TVD Runge-Kutta time integrators

Explicit Runge-Kutta (ERK) time integration schemes preserve the TVD character of a spatial discretization method, if they are convex with respect to their intermediate

steps and are at least of order *p+1* with respect to order of the spatial discretization method *p*. Hence, we implemented higher order TVD-Runge-Kutta schemes up to third order.

However, the usage of a TVD-Runge-Kutta scheme is not sufficient to assure TVD. In addition, it is necessary to impose a limiting method to ensure the TVD property for the spatial discretization. This can be provided by either limiting the flux through the cell surfaces and/or by limiting the higher order coefficient moments.

4.4.2. Limiter methods

For piecewise constant finite elements, i.e. Legendre polynomials of order *p=0,* the DG-FEM scheme is identical to the finite differencing scheme described in Sec. 4.2. Hence, the finite differencing scheme was used to evaluate the DG-FEM for the special case *p=0* with and without flux limiter.

Several different types of limiter approaches were implemented and their influence on the numerical solution of the 1D advection equation was investigated. These are namely a generalized flux limiter approach, a TVDM-slope-limiter, and a simple and a generalized moment limiter. The first limiting method is acting only onto the flux terms of the scheme, whereas the latter three methods are limiting the whole coefficient state vector.

The flux limiter approach is not capable of eliminating spurious oscillations in the discontinuous solution of the advection equation when using higher order (*p>0)* Runge-Kutta-DG-FEM schemes. The blended flux provided by this method has in either case a lower or equal viscosity than/as the “upwind” flux. Hence, wiggles will be less efficiently suppressed compared to the standard DG method using exclusively the upwind flux.

The TVDM-slope-limiter, although assuring the TVD property of the scheme for the cell averaged values, i.e. the Legendre coefficients for the piecewise constant finite elements, however, generates some severe artefacts in the numerical solution. Local extrema within smooth parts of the solution are suffering from clipping and distortion.

The two moment limiters yield better results than the TVDM-slope-limiter. But schemes using the simple limiter are only total variation bound (TVB) and suffer from clipping and distortion as well, however, less severe. Furthermore, in case of the generalized moment limiter there exists no mathematical proof for its TVB property. Anyway, our results clearly show that the generalized moment limiter provides the most promising results of all tested limiters (see the attached technical report). The limited solution does not suffer from clipping at all. Moreover, due to the flexible upper bound of limiting this limiter can preserve finer details of the solution, especially when combined with higher order methods (*p>1*).

4.4.3. Implicit Runge-Kutta time integrators

The ERK time integration schemes from Sec. 4.4.1 need a quite restrictive time step ($\Delta t<0.1$) for well resolved numerical results. Hence, it was of interest to further investigate some implicit time integrators for the DG-FEM method. As part of their implementation the explicit solver had to be rewritten in a matrix-vector form. This was necessary to provide the corresponding matrix inversions for implicit schemes by a direct solver package (LAPACK). Similar to the finite difference schemes from Sec. 4.2 a whole family of time integration schemes has been provided, e.g. an explicit scheme, a Crank-Nicholson scheme, a fully implicit scheme (Eulerian backward) and higher order Runge-Kutta implicit time discretization schemes.

The implemented implicit Runge-Kutta (IRK) multistage schemes are of different type: Gauss-Legendre IRK schemes up to four stages and eighth order accuracy, IRK schemes of Radau type up to fifth order and finally two fourth order accurate singly-diagonally-implicit Runge-Kutta (SDIRK) schemes. First, the IRK schemes were tested by solving a stiff ordinary differential equation as a test problem. Afterwards, all the IRK schemes were tested with our standard advection equation test problem used already for the explicit solver. In contrast to the Eulerian backward scheme, the various implemented IRK schemes turned out to be not unconditionally stable for arbitrarily large time steps. All of them showed, with increasing time step growing spurious oscillations. These wiggles could be controlled to a certain extend with the moment limiter methods introduced in Sec. 4.4.2. However, the final instability of the scheme for large time steps could not be prevented but only shifted to larger time steps.

4.4.4. Conclusions for DG-FEM Runge-Kutta schemes

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 due to the iterative solution of the coupled time integration levels by using a fixed point iteration. Hence, the ability to use larger time steps for IRK methods compared to ERK methods did not pay off in the overall efficiency.

4.5. Discontinuous Galerkin scheme for the 2-dim model problem

After successfully implementing several DG-FEM schemes for the linear 1-dim advection equation we continued with the (1+1) dimensional model problem specified by the project coordinator, W. Arter. It is based on the PhD thesis of A. Belli and is essentially an advection problem with a distribution function in the coordinate's space *s* and velocity *v* plus an integral source term over velocity space.

In a first step, the explicit DG-FEM solver described above was extended to solve simultaneously the advection equation for a number of varying advection velocities both of negative and positive values. Accordingly, a velocity mesh was defined to extend the model into the second dimension *v.* In the next step, the integral source term over velocity space was written in a form to merge it with the spatial advection term. For its implementation it was crucial that the DG-FEM scheme is closely related to the finite volume method. This reduced the integration over velocity to a simple sum over the coefficient vectors in the direction of velocity.

The implemented (1+1) dimensional scheme was carefully tested with different initial conditions to check its operating ability. Finally, for further testing with physical parameter sets the program was delivered to W. Arter.

4.6. Dissemination of OPTGS2 project

In collaboration with W. Arter, an abstract for the 21st International Conference on Numerical Simulation of Plasmas 2009 (ICNSP 09) was submitted (http://icnsp09.ist.utl.pt/code/preview.php?abs_id=32). The talk was given by W. Arter on 8th October at the ICNSP 09 in Lisbon.

Finally, Nicolay Hammer visited the project coordinator, W. Arter and collaborators in the fusion department of Culham Science Centre. The one week trip took place from 30th November to 4th December. An informal talk on the project's topic was given and

several meetings with scientists being involved in the project took place. All programs written so far have been delivered to W. Arter and explained in detail.

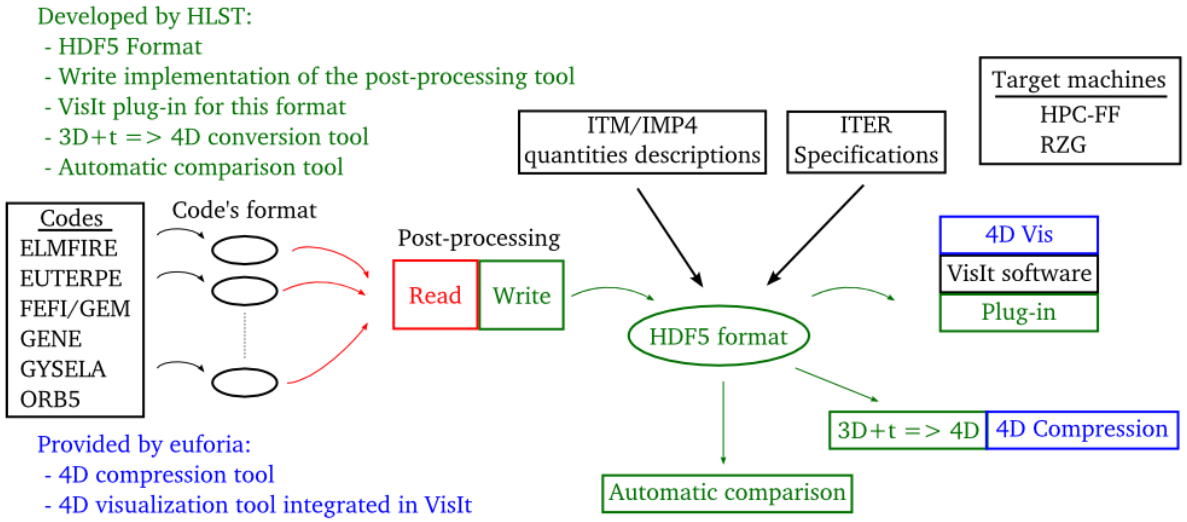
5. Proposed GYNVIZ project

5.1. Abstract

The present project is the result of a request for graphical support from several European gyrokinetic turbulence simulation groups in the last HLST call. Instead of targeting on individual solutions, the proposed project offers a general solution by involving the principal investigators of the major European gyrokinetic simulation codes. It addresses the general issue of data post-processing and visualization for the gyrokinetic codes' outputs. It is structured in three main parts. First, it aims at merging and extending the two initiatives of ITM and ITER dedicated to the design and the implementation of a unified format for gyrokinetic codes' outputs. Based on this format, the second part consists of providing a consistent and unified set of visualization tools to answer the needs of the involved code developers. Finally, we propose to settle on a remote visualization framework to provide centrally managed and high-end visualization resources for these new tools without additional hardware costs for the different simulation groups.

5.2. Detailed project description

The aim of this project is to gather the requests concerning visualization support from different European gyrokinetic turbulence simulation groups. The main idea is to provide a consistent and unified set of tools to the entire community.

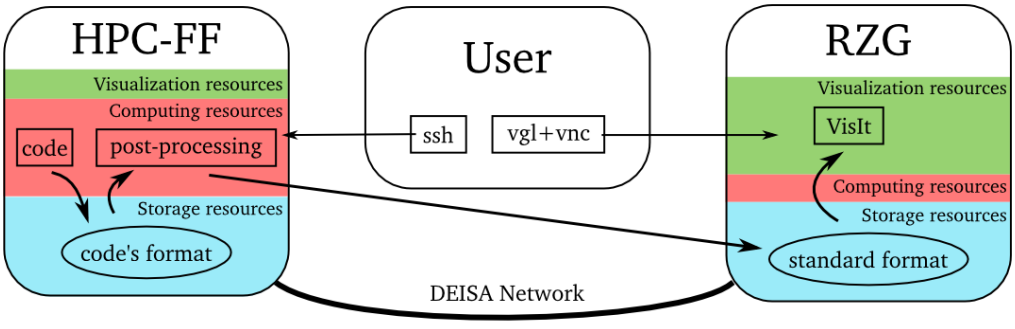


As shown on the project scheme, the central concern is the design and the implementation of a uniform data format supported by all gyrokinetic codes. This work has already been started within the Integrated Modelling Project 4 (IMP4) of the Integrated Tokamak Modelling (ITM) and ITER. On the one hand, ITM/IMP4 initiative is more focused on standardizing how physical quantities are actually computed. On the other hand, ITER specifications focus rather on how the data is organized from a numerical point of view. The present project aims at merging and extending these two initiatives by an actual format implementation. As the real usage of this format in production is of great importance to our point of view, we will design this format as flexible as possible in order to let the physicists enough freedom to experiment with new issues. So a trade-off between standardization and flexibility has to be found. We propose to adopt an incremental approach, by releasing different versions of the format over the project duration. This would allow the implementation of first simple datasets while ongoing effort is spent to design a format for more complex datasets.

Based on such a format release, we propose to develop a set of interoperable tools. At the stage of the proposal, we have at least three objectives. Firstly, an automatic comparison tool is straightforward to develop as soon as the data are computed and stored in the new uniform format. Secondly, we propose to introduce the use of the latest open source 3D visualization software. The development of an XDMF[1] format and/or a database reader plug-in for such software will enable the user to directly display the data. VisIt[2] is our first target software because it is currently one of the most prevalent open source 3D visualization software. It is developed and durably maintained by Lawrence Livermore Lab in the United States. Finally, we propose to get 4D compression and visualization tools developed within the EUFORIA project and to make them available to this community. The 4D compression consists basically in projecting the 4D function in hierarchical bases of finite elements. The 4D visualization consists in building interactively 2D slices from the compressed representation of the function, so the user can browse the 4D space by looking at refreshing 2D slices. We will also extend these tools by developing a program that turns 3D time-varying data into 4D data. Then, using the 4D visualization, one will be able to follow a particular structure in space and time dimensions at one time.

The actual generation of files in the new unified format is a matter that has to be addressed and it will be partly supported by the HLST. Indeed, the different numerical methods used by the different codes imply naturally different internal data structures and consequently different ways of exporting data. The HLST will not provide a data converter for every code. Instead it will provide a Fortran 90 program that allocates arrays in memory and writes their content in a file with the right format. It will be up to the code developers to implement either the “Read” part of this program or directly integrate these “Write” subroutines into their codes.

In addition, the visualization resources for these new tools are an issue to be addressed. Currently, they are very often local to the end user and not very powerful. Even worse, it is sometimes the user who has to install and maintain the needed visualization software. We propose to move the present scheme to the following one which includes remote visualization. It means that the memory and compute intensive parts of a rendering task execute on specialized graphics hardware operated in the "cold room" of a computing centre. Specific software accounts for the efficient transmission of the graphical display over the network and handles the communication of user interaction. This completely eliminates the need for high-end graphics workstations on the client side, i.e. researchers can perform large-scale visualization tasks interactively on ordinary notebooks or PCs across wide-area networks.



The figure above depicts the first implementation which should be available in summer 2010 as a demonstrator. It implies the HPC-FF machine in Jülich which is already in production and a graphic cluster dedicated for remote visualization which should be built during spring 2010 at the Rechenzentrum Garching (RZG). It will

[1] www.xdmf.org
[2] https://wci.llnl.gov/codes/visit

consist of 4-8 servers each with at least 8 cores and 128 GB RAM. The cluster will embed 8 GPUs and a dedicated GPFS file system (~100 TB) visible to RZG's major HPC file systems. Thanks to the 10 Gb/s DEISA network[3] between JSC and RZG, users will have a transparent access to their data.

To bring the participating simulation groups to a consistent level of knowledge in using the new visualization environment, training and support will be provided on the various aspects of the project.

As a conclusion, this projects aims at designing and implementing a unified format for gyrokinetic codes' outputs. Based on this format, a consistent and unified set of visualization tools will be provided. By using the graphic hardware of RZG it will be possible to provide to the group members access to a remote visualization framework.

[3] www.deisa.eu/services/infrastructure