D6.1.3 – Roadmap to exascale (update 2)

WP6: Co-design via applications

Project Acronym	CRESTA
Project Title	Collaborative Research Into Exascale Systemware, Tools and Applications
Project Number	287703
Instrument	Collaborative project
Thematic Priority	ICT-2011.9.13 Exa-scale computing, software and simulation

Due date:	M39
Submission date:	01/12/2014
Project start date:	01/10/2011
Project duration:	39 months
Deliverable lead organization	CSC
Version:	1.0
Status	Final
Author(s):	Mark Abraham (KTH), Mikko Byckling (CSC), Willem Deconinck (ECMWF), Derek Groen (UCL), Jing Gong (KTH), Baerbel Grosse-Woehrmann (HLRS), Mats Hamrud (ECMWF), Timo Krappel (USTUTT), George Mozdzynski (ECMWF), Adam Peplinski (KTH), Florian Seybold (HLRS), Jan Åström (CSC), Jan Westerholm (ABO)
Reviewer(s)	Alan Gray (EPCC), Markus Flatken (DLR), Lorna Smith (EPCC)

Dissemination level	
<PU/PP/RE/CO>	*PU*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	29/08/2014	Template of the deliverable	Mikko Byckling (CSC), Jan Åström (CSC)
0.2	26/11/2014	Added introduction and summary	Mikko Byckling (CSC)
0.3	26/11/2014	IFS contribution	George Mozdzynski (ECMWF), Mats Hamrud (ECMWF) , Willem Deconinck (ECMWF)
0.4	27/11/2014	HemeLB contribution	Derek Groen (UCL)
0.5	28/11/2014	OpenFOAM contribution	Baerbel Grosse-Woehrmann (HLRS), Timo Krappel (USTUTT), Florian Seybold (HLRS)
0.6	28/11/2014	Nek5000 contribution	Jing Gong (KTH), Adam Peplinski (KTH)
0.7	1/12/2014	Minor corrections	Mikko Byckling (CSC)
0.8	2/12/2014	Gromacs contribution	Mark Abraham (KTH)
0.9	2/12/2014	Elmfire contribution	Jan Westerholm (ABO)
1.0	2/12/2014	First version for internal review	Mikko Byckling (CSC)
1.1	9/12/2014	Addressed reviewer comments	Mikko Byckling (CSC)
1.2	10/12/2014	Addressed reviewer comments	Mikko Byckling (CSC)
1.3	11/12/2014	Addressed reviewer comments	Mikko Byckling (CSC)
1.4	17/12/2014	Addressed reviewer comments	Mikko Byckling (CSC), Lorna Smith (EPCC), Jan Westerholm (ABO)
1.5	17/12/2014	Final version for EC review	Mikko Byckling (CSC)

Table of Contents

Index of Figures

Index of Tables

1 Executive summary

This document contains an update to the two initial roadmaps for the CRESTA codes described in Deliverables D6.1.1 and D6.1.2. Description of the main developments and application performance improvements conducted during the project, as well as an update to the original roadmaps for the separate codes, are summarized in Section 1.1. Activities related to the co-design progress are summarized in Section 1.2 for each application separately.

1.1 Summary of progress and roadmap to exascale

The main progress and roadmap to exascale for each application can be summarized as follows:

ELMFIRE: Memory consumption has been reduced during the project. Roadmap to exascale consists of further reducing the memory consumption and finishing the implementation of the 3D domain decomposition.

GROMACS: The performance has improved during the project. In addition, an ensemble framework has been implemented to perform massively parallel ensemble computations consisting of many large parallel simulations. Roadmap to exascale targets hybrid task parallelism, algorithmic improvements for PME electrostatics and enhanced use of accelerators in the computations.

HemeLB: The performance and scalability of the code has improved significantly during the project. In addition, to improve the visualization and computational steering capability of the code, tools from CRESTA WP5 have been integrated into the code. Roadmap to exascale targets code optimization towards many-core architectures and the use of ensemble simulations.

IFS: The performance and scalability of the code has improved significantly during the project. Several preliminary investigations towards the roadmap to exascale have been performed, resulting in a robust plan to have an implementation of IFS for exascale via the use of GPGPU, task graph scheduling and a partially revised codebase.

NEK5000: The code has been enabled to use GPU devices via OpenACC accelerator constructs. The achieved performance of the multi-GPU implementation is slightly better than a similar number of CPU nodes. Implementation of adaptive mesh refinement enables automated refinement of areas of interest in the computation. Roadmap to exascale focuses on an improvement of multi-GPU communication and an implementation of an efficient pressure preconditioner for adaptive mesh refinement.

OpenFOAM: Development ceased within CRESTA after M24. Large disruptive changes are required in order for OpenFOAM to use exascale-level hardware.

1.2 Summary of co-design activities

The co-design activities for each application are summarized below. For experiences of the co-design process, see Section 2.1.

ELMFIRE: ABO participated in Lattice Boltzmann on GPUs co-design with HemeLB and OpenACC co-design and performance evaluation with Cray. In addition, visualization of simulation results was implemented to ELMFIRE as a co-design effort with CRESTA WP5.

GROMACS: GROMACS participated in 3DFFT co-design as well as nonblocking collective co-design with CRESTA WP4. Additionally, as an OpenACC co-design effort with Cray, a performance comparison between the hand-written CUDA kernels from GROMACS with compiler generated OpenACC kernels was accomplished.

HemeLB: HemeLB participated in Lattice Boltzmann on GPUs co-design with ABO, co-design related to pre –and postprocessing with WP5, and nonblocking collective co-design with WP4.

IFS: IFS participated in Fortran coarrays co-design with CRAY and development environment co-design with TUD. Task graph co-design work on OmpSs was done in collaboration with European exascale projects DEEP and MB.

NEK5000: Nek5000 participated in global communication and computational kernel autotuning co-design with WP3 as well as OpenACC co-design with Cray.

OpenFOAM: OpenFOAM participated in numerical libraries co-design with WP4.

2 Introduction

This document contains roadmaps detailing the actions needed to further develop the CRESTA codes towards exascale performance after the end of the project.

For all applications, except OpenFOAM, the document also illustrates the performance and scalability improvements achieved during the course of the project. For OpenFOAM, an analysis is presented on why the code is not an exascale-capable code and what kind of design principles should be applied should a code with a similar functionality be developed.

The roadmaps and results for the different applications are presented in the following chapters. The functionality and research goals of the applications can be summarized as follows:

ELMFIRE: is a gyro kinetic particle-in-cell code that simulates movement and interaction between high-speed particles in a torus-shaped geometry on a three-dimensional grid. The particles are held together by an external magnetic field. The objective is to simulate significant portions of large-scale fusion reactors like JET or ITER.

GROMACS: is a molecular dynamics code which is extensively used for simulation of biomolecular systems. Useful investigation of this kind of system is typically limited by computational capacity. The limitations relate both to system sizes and in particular time duration of the processes of interest. Efficient implementation of ensembles of simulation are also needed in order to obtain statistical validity.

HemeLB: is intended to form part of a clinically-deployed exascale virtual physiological human. HemeLB simulates blood flow in measured blood vessel geometries. The objective is to develop a clinically useful exascale tool.

IFS: is the production weather forecasting application used at the European Centre for Medium Range Weather Forecasts (ECMWF). The objective is to develop more reliable 10-day weather forecasts which can be run in an hour or less.

NEK5000: is an open-source code for the simulation of incompressible flow in complex geometries. Simulation of turbulent flow is of one of the major objectives of NEK5000.

OpenFOAM®: is an open source application for computational fluid dynamics (CFD). The program is a “toolbox” which provides a selection of different solvers as well as routines for various kinds of analysis, pre- and post-processing. Besides general development of the code, within this project the focus will be on a specialized code for turbo machinery. The objective is to simulate a whole hydraulic machine on exascale architectures.

2.1 Co-design process

Co-design has been used for decades to aid software and hardware design [1]. One of the main questions at the beginning of the project was whether the co-design idea can be made to work. To answer the question in short: yes, the co-design process as it was implemented in the project was clearly beneficial to the CRESTA applications. For similar discussion with a broader focus on how the co-design process can be made to work, see [2].

The obtained benefits of co-design differed somewhat between the applications. Also, in order to have a less unequivocal picture, one needs to consider the opposite, i.e., what were the benefits of the co-design applications to the other parts of the project? Both of these viewpoints are discussed in the following two subsections, albeit with somewhat more focus on the benefit to the applications themselves.

2.1.1 Application development and the co-design process

In this subsection, we highlight some of the cases where application development benefited from the co-design collaboration.

HemeLB, computational steering, domain decomposition and visualization

With WP5, an improved toolkit for visualization and computational steering was implemented to replace the old visualization toolset in HemeLB. In addition, to improve the partitioning, domain decomposition libraries from WP5 were used to implement a prototype of a partitioner for sparse geometries.

IFS, Fortran coarrays

With Cray, overlapping of communication and computation in the context of OpenMP parallel regions was implemented in IFS. As using Fortran coarrays within OpenMP had not been attempted before in a production application, close collaboration with Cray was essential both in the design phase as well as in the implementation process itself.

Nek5000, autotuning of computational kernels

With WP3, autotuning work on key computational kernels was performed. The new autotuned kernels improve the computational performance over the previously used handwritten kernels.

2.1.2 Applications as co-design process vehicles

In this subsection, we highlight some of the cases where co-design applications had a clear benefit to the systemware developers.

Cray, comparison of GROMACS kernels

GROMACS developers provided their handwritten CUDA compute kernels to Cray for comparison with functionally identical kernels written with OpenACC. The work provided improvements to the register usage of the Cray OpenACC compiler, improving the performance of the compiler-generated code.

WP5, autotuning of Nek5000 computational kernels

Nek5000 developers provided their key compute kernels to WP3 for autotuning work. The kernels provided a realistic test bed for autotuning tools developed in WP3 and showed that using computer generated parameter searches can provide actual performance benefits in production applications.

2.1.3 CRESTA application developer perspective to co-design process

In this subsection, we describe from an application developer perspective how the co-design process was implemented within CRESTA.

For application developers in CRESTA WP6, establishing a functional co-design collaboration with another member of the project was, at least in some cases, a relatively long process. Formation of a co-design team was done as follows:

1. Establish contact with the other members of the co-design team
2. Make a collaboration plan
3. Formally establish the co-design team

Usually the co-design work was initialized via personal contacts established during CRESTA collaboration meetings. The key issue was to get the developers to know the problem area the potential candidates for collaboration were working upon. Establishing personal contacts via collaboration meetings was an effective but not necessarily the fastest method for building the co-design teams, as people new to the project were mostly unknown outside of their own work package until the next, typically biyearly, collaboration meeting had been held. Further details of the work were then agreed upon after the meeting via email, mailing lists, conference calls or even personal visits. This included the construction of a plan on how to progress with the newly defined co-design task.

Finally, whenever deemed appropriate, a co-design team was formally formed to handle the co-design task. Most of the communication was established via continuous personal contacts with the involved parties. Personal visits and small meetings were also used to work together and better agree upon outstanding issues, although, due to geographical separation of the project members and the associated travelling overhead, such collaboration methods were used rather sparingly.

The results of the co-design teams were presented during CRESTA collaboration meetings and, during later stages of the project, via CRESTA white papers. Some of the presentations led to co-design work being extended to include more partners to benefit from the results. In some cases, when the task of the co-design team was finished, the team disbanded as being no longer necessary.

2.1.4 Conclusions on the co-design process

When interviewed at the end of the project, all application developers who had been involved in co-design considered the experience very positive. Many commented that having domain experts from another field significantly sped up the development process and cut down implementation time. For instance, people involved in the Fortran coarrays co-design effort argued that the development would have been much more difficult (or even near to impossible) without the help of the Cray experts.

In CRESTA, the start-up cost to have a productive co-design effort was non-negligible. Thus avoiding projects which are too short in duration is recommended in order to avoid unnecessary overhead. In order to speed up the formation of the co-design teams with a mixed base of people, we recommend establishing an easily accessible list which describes the expertise, contact details and co-design efforts in which each person within the project is involved. Organizing several co-design meetings early on in the project to establish personal communication channels is highly recommended as well.

Mini applications mimicking the behavior of the full application were used in CRESTA to study whether or not and idea was worth pursuing in an actual application. Relying solely on mini applications is not recommended, however, as it may not give an accurate view of the difficulties faced when implementing changes to a real world production code. Such difficulties were demonstrated in CRESTA by the rather straightforward use of OpenACC directives to port the Nekbone benchmark to use GPGPUs. This was followed by a rather long but eventually successful process of porting the actual Nek5000 application to GPGPUs with OpenACC.

To conclude discussion on co-design within CRESTA, we state that the co-design idea was a powerful tool for efficiently conducting cross-disciplinary work. According to the experiences gathered by application developers, we recommend the use of co-design as a tool in other exascale software efforts as well.

2.2 References

[1] Sanders, E. B. N. and Stappers, P. J. Co-creation and the new landscapes of design. Co-design, 4(1), 5-18. 2008.

[2] Dongarra, J. et al. The international exascale software project roadmap. International Journal of High Performance Computing Applications, 1094342010391989. 2008.

2.3 Glossary of Acronyms

ACML	AMD Core Math Library
AMI	Arbitrary Mesh Interface
AMR	Adaptive Mesh Refinement
CAF	Coarray Fortran
CSC	CSC – IT Center for Science Ltd.
CPU	Central Processing Unit
DLR	Deutschen Zentrums für Luft- und Raumfahrt
ECMWF	European Centre for Medium-Range Weather Forecasts

ECSE	Embedded Computer Software Engineering
ENDA	Ensemble Data Assimilation System
EPCC	Edinburgh Parallel Computing Centre
EPS	Ensemble Prediction System
FFT	Fast Fourier Transform
GGI	General Graphics Interface
GNU	GNU's Not Unix!
GPL	GNU General Public License
GPU	Graphics Processing Unit
HPC	High Performance Computing
INCITE	Innovative and Novel Computational Impact on Theory and Experiment
I/O	Input/Output
ITER	International Thermonuclear Experimental Reactor
JET	Joint European Torus
KTH	Kungliga Tekniska Högskolan
LB	Lattice Boltzmann
LGPL	GNU Lesser General Public License
MPI	Message Passing Interface
OpenACC	Open Accelerators
OpenMP	Open Multiprocessing
PETc	Portable, Extensible Toolkit for Scientific Computation
PGAS	Partitioned Global Address Space
PME	Particle Mesh Ewald
SIMD	Single Instruction, Multiple Data
UCL	University College London
USTUTT	University of Stuttgart

3 Elmfire

Elmfire is a particle-in-cell code that simulates the movement and interaction between extended gyrokinetic particles moving at high speed in a torus-shaped geometry on a three dimensional grid. The particles are held together by a strong external magnetic field.

Elmfire approximates the Coulomb interaction between particles by solving a global electrostatic field on a grid, using the particle charges as sources. Elmfire then advances particles in time by free streaming along the magnetic field line and particle drift perpendicular to the magnetic field. Typically, time steps correspond to 30-50ns of real time.

The time step based simulation in Elmfire can be roughly divided into seven parts:

- Perform collisions between particles close to each other;
- Using a 4th order Runge-Kutta, calculate particle movements in continuous space during the time step based on the electric field;
- Collect grid cell charge data from the particles for the electrostatic field;
- Combine and split the grid charge data so each processor has a smaller part of it;
- Construct a large modified gyro kinetic Poisson equation based on the data and solve it in parallel;
- Calculate additional movement caused by magnetic field drift of particles based on the acquired electric field;
- Write diagnostics output.

Presently, the most CPU-heavy part of the code is calculating particle movements, but as each processor is assigned a fixed number of particles this scales linearly with the number of processors and is therefore not the most significant issue when scaling to larger systems. The most problematic part is the collection and distribution of grid cell charge data. In the current version each processor can have its assigned particles moving in any part of the torus, leading to all processors contributing charge data to all grid cells in the system. As a consequence each processor has the full electrostatic grid data and a huge sparse matrix, the size of which is the number of grid cells squared, for collecting charge data. To place this in context, simulating ITER with this version of the code would require 640K cores, each with 28TB of memory.

To reduce the memory consumption, a new domain decomposition algorithm has been written for Elmfire within the CRESTA project. This has been a significant re-design as it has involved changes to almost all the components of Elmfire. Within this, the simulation volume is divided among cores to harmonize with the equations of motion. The domain decomposition is in line with high particle speeds in the toroidal direction. The result is a diminished memory usage per core, as the simulation volume considered by one core has been reduced drastically.

Within Elmfire all the cores collaborate to solve the Poisson equation and further optimization has been achieved through a co-design process with WP4 to implement a modified version of the Poisson equation, to further enhance the memory utilization. Once the grid cell charge data has been combined and split among the processors, each processor can construct its own part of the Poisson equation individually. The Poisson equation is then solved in parallel using PETSc. The solution (the electric potential) is then distributed to all processors to be used in the next time step.

While further memory improvements can and will be made in the future, the implementation of a domain decomposition version of Elmfire has significantly decreased the memory utilization.

3.1 Summary of the previous roadmaps

Task	Scheduled date	Status
3D domain decomposition	M22	Completed
Processor load balancing	M28	Completed
Memory usage for binary collisions	M36	Completed
Parallel file writing	M36	Completed

Table 3.1 Summary of the previous roadmaps for Elmfire

3D domain decomposition

Prior to CRESTA, the code implemented as 1D decomposition in the toroidal direction with a full electrostatic grid on each processor, with all processors having particles in any given grid cell. During CRESTA we have implemented a 3D domain decomposition where each processor owns a subgrid of the electrostatic grid and all particles within these grids. This diminishes the memory consumption of each processor which otherwise would grow with grid size. The domain decomposition is streamlined with the time evolution of the particles in the sense that very large particle velocities are common in the toroidal direction (contained fully in the domain of one core) while velocities are much smaller in the cross sectional directions (particles may be handed over to cores owning neighboring grid volumes). At the same time each processor needs only a small part of the electrostatic grid close to its own volume in order to be able to propagate particles during one time step.

Processor load balancing

Each processor is allocated approximately the same number of particles according to the initial assumed particle distribution, denser in the centre, fewer particles at the outer surface, by allocating full circles of subgrids in the toroidal direction.

Memory usage for binary collisions

Previously Elmfire communicated particle locations in order to simulate particle collisions. In the new domain decomposition, processors automatically contain all particles in any given grid cell and thus binary collisions can be simulated with no communication.

Parallel file writing

Within CRESTA, Elmfire has been written to use hdf5 file I/O for dumping aggregated data (densities, energies, etc). Future plans beyond CRESTA are to include snap-shotting particles in Elmfire for restart purposes using parallel file I/O. This is planned for the year 2015.

3.2 Achievements towards remaining tasks

Task	Achievement
Parallel file writing	All file I/O now use the hdf5 library

Table 3.2 Achievements towards remaining tasks from the previous roadmaps for Elmfire

3.3 Roadmap to exascale

Task	Estimated effort	Status
Rewriting memory allocation and structure for the Poisson equation	M18	Ongoing

Table 3.3 Roadmap to exascale for Elmfire

3.3.1 Rewriting memory allocation and structure for Poisson equation

Prior to CRESTA, each processor allocated memory for the full Poisson grid. The domain decomposition developed within CRESTA ensures that each processor owns a small part of the Poisson grid. Future work will consider additional optimizations to further reduce the amount of allocated memory per processor.

3.4 Application performance and scalability

Performance runs have been carried out with the new domain decomposition code on half a billion particles on up to 4096 cores on Sisu (Cray XC40). The numbers below describe weak scaling where we increase the number of grid points while keeping the number of particles per grid point constant for load balancing reasons. From a physical point of view the increase in grid size corresponds to larger tokamak simulation volumes, the largest being 256x256 grid points in the cross section and 16 grid points along the toroidal direction corresponding roughly to a fusion reactor with cross section diameter of 1 m.

Cores	Grid size	# of particles	# of particles/core
128	32x64x16	14329856	111952
256	64x64x16	26005504	101584
512	64x128x16	52692992	102916
1024	128x128x16	107876352	105348
2048	128x256x16	206671872	100914
4096	256x256x16	444991488	108640

These results demonstrate that memory consumption per core is currently almost proportional to the number of particles, a significant achievement and an original objective set within CRESTA.

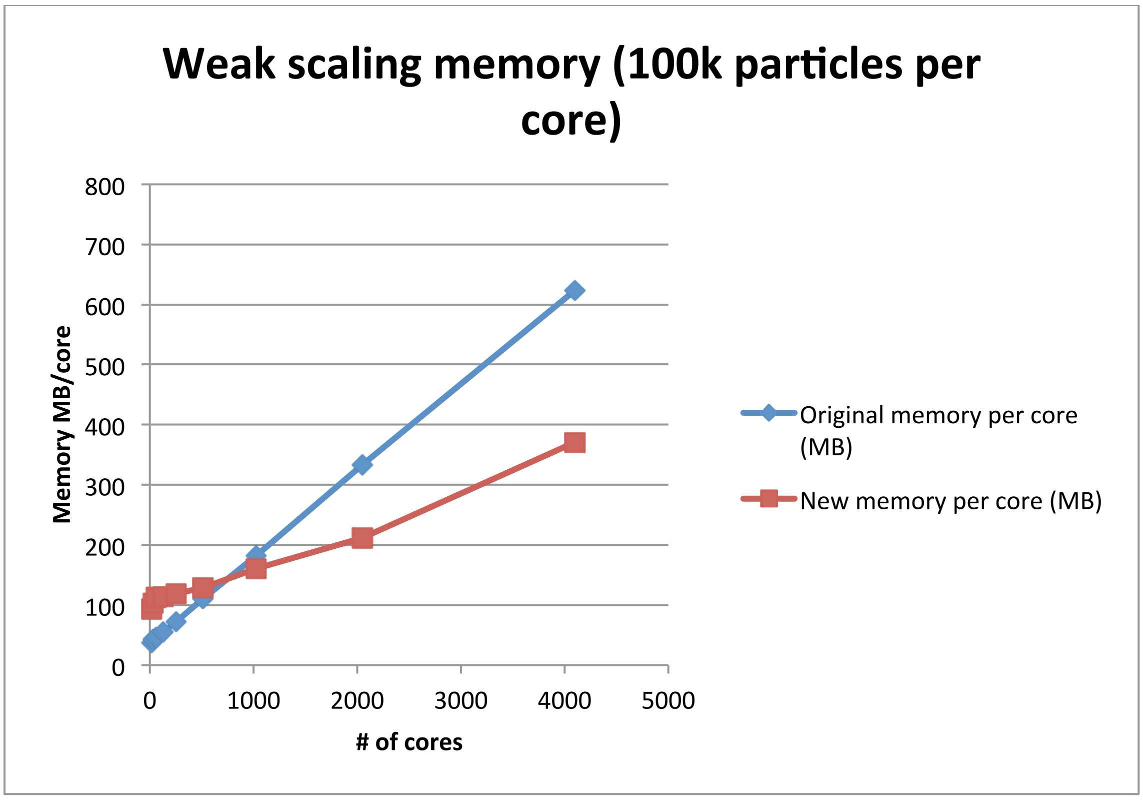


Figure 3.1 Elmfire memory scalability per core in a weak scaling test for a model problem

Beyond CRESTA, future performance studies will look at even larger systems. In particular, during 2015 the plan is to investigate particle collisions.

3.4.1 Summary of the performance improvements achieved

The implementation of a 3D domain decomposition has resulted in improved memory consumption and memory organization, and hence better scalability. This was the main objective of the work within CRESTA as memory consumption was a significant block towards scaling this code to future (exascale) systems. In addition co-design optimization work with WP4 on the linear solver has improved the code's overall performance. Elmfire has been written to use hdf5 file I/O for dumping aggregated data (densities, energies, etc). Finally, the original code was a "patch-work" of added features, and significant work has gone into rewriting the code for better readability and structure.

3.5 References

[1] Exemplar scientific simulations, CRESTA Deliverable D6.4.

4 Gromacs

GROMACS is a major open source code that performs classical molecular dynamics simulations based on interactions between particles moving in space, typically for biomolecular systems. It has been developed for over 15 years, initially with a large focus on the highest possible single-core performance. Over the last few years we have made a complete overhaul of the parallelization approach and the code currently exhibits some of the best relative scalability in the field.

The main challenge for classical molecular dynamics in general - and GROMACS in particular - is that it relies on integration of Newton's equations of motion, and high performance therefore requires very fast iterations over integration time-steps. This has largely driven the past 20 years of development and thus the current algorithms are focused on providing simple interaction forms to reduce the floating-point instruction bottleneck.

Historically, runtime for codes performing molecular dynamics was completely dominated by the evaluation of interactions between particles and, at least in principle, this lends itself very well to parallelization. Unfortunately the last 20 years of optimization focused on algorithms to avoid floating-point operations has resulted in complex data structures and inhomogeneity in interaction density over space that makes efficient parallelization challenging. In this regard, GROMACS is a particular challenge since the single-core performance is significantly higher than many other codes and thus the time spent on communication is relatively larger [5].

The work in GROMACS focuses on achieving significant improvements for real applications. From the end user perspective, there are three overall important objectives to advance the state-of-the-art for applications

(i) Reduce the wall-clock time per time-step of iteration in order to achieve longer simulations.
(ii) Handle much larger application systems to model e.g. mesoscopic phenomena.
(iii) Improve the accuracy and the results for small application systems through massive sampling.

All three aspects are critically important, but require slightly different approaches. The wall-clock time for a single time-step iteration is already today in the range of a few milliseconds on some systems, and even though strategies exist to improve this further, we do not believe that it is possible to push more than one order of magnitude beyond today's standard. In contrast, handling much larger systems is easier (although not trivial) from the point of view of a parallelization algorithm. Unfortunately it will involve challenges related to handling of data when a single master node can no longer control all of the input and output, both when starting the execution and when performing checkpointing or output. Finally, for small systems, the main approach will be the use of ensemble techniques to handle thousands of small simulations each of which will use thousands of cores.

4.1 Summary of the previous roadmaps

Task	**Scheduled date**	**Status**
Benchmarking new Gromacs releases, and GPU coding	M18, M30	Completed
Multi-grid solvers for efficient PME electrostatics	M36	Ongoing
Task-based parallelism	M36	Ongoing
Efficient large-scale I/O	M36	Ongoing
Ensemble computing & parallel adaptive molecular dynamics	M36	Completed

Table 4.1 Summary of the previous roadmaps for Gromacs

Benchmarking new Gromacs releases, and GPU coding

The Gromacs 5 release was completed, and has been benchmarked for use in upcoming publications. The CRESTA benchmark suite was updated to use the features implemented during the CRESTA project. Through our ongoing collaboration with NVIDIA, CUDA features unveiled at SC'14 were already supported in the Gromacs development branch.

Multi-grid solvers for efficient PME electrostatics

Changes to Gromacs to be able to use ExaFMM (http://www.bu.edu/exafmm/) for multipole-based long-range electrostatics are underway. The dominant implementation of full electrostatics used by domain scientists has been the particle-mesh Ewald method (PME), and the code in Gromacs for full electrostatics treatments needs extension and generalization to permit multiple implementations to co-exist.

Task-based parallelism

Extensive preparations for full-scale conversion to task parallelism are in progress. We have identified Intel's Thread Building Blocks (http://www.threadbuildingblocks.org) as the tasking framework most likely to deliver performance portability in Gromacs. It is implemented as a C++98 library, has a permissive source-code license, and seems likely to provide control at sufficiently fine grain to keep per-task overheads to at most around 1 microsecond (required for improving strong scaling with Gromacs). We have submitted numerous bug reports to Intel to improve the support in TBB with non-Intel compilers.

Efficient large-scale I/O

The exascale-suitable implementation of the TNG compressed output-file format [7][8] was completed in Gromacs 5. Like other I/O implementations in Gromacs, it currently runs in serial. This is not yet a critical problem to solve, since typically only a small fraction of the atoms in the system are of interest and the period with which statistically independent output is available is only every thousand or more MD steps. It has been planned for the I/O code to run in parallel as a non-blocking task in the new tasking framework.

Ensemble computing & parallel adaptive molecular dynamics

Copernicus 2.0 has been released, [3], and is in use in production simulations. For instance, by using Copernicus we have been able to scale a protein folding problem to 5730 cores – reducing time-to-solution from 30 days to 72 hours. Copernicus currently includes implementations of the following adaptive sampling algorithms: Markov state modelling, adaptive free energy perturbation and a string method for minimum free energy pathways. Each of these typically scales to hundreds or thousands of simulations in parallel, each of which itself can be parallelized to tens to hundreds of cores, making simulations with more than 1M cores feasible.

4.2 Achievements towards remaining tasks

Task	**Achievement**
Multi-grid solvers for efficient PME electrostatics	Early implementation using ExaFMM for full electrostatics.
Task-based parallelism	Planning, core infrastructure changes.
Efficient large-scale I/O	Implementation of the actual efficient I/O.
Improved domain-decomposition halo exchange	First implementation completed and under testing.

Table 4.2 Achievements towards remaining tasks from the previous roadmaps for Gromacs

4.3 Roadmap to exascale

Task	Estimated effort	Status
New decomposition for bonded interactions	1 PM	Code in progress
Task-based parallelism	18 PM	Ongoing
Efficient large-scale I/O	1 PM	Ongoing, depends on task-based parallelism
Improved domain-decomposition halo exchange	1 PM	Under testing
Multi-grid solvers for efficient PME electrostatics	6 PM	Ongoing

Table 4.3 Roadmap to exascale for Gromacs

4.3.1 New decomposition for bonded interactions

The existing dynamic load balancing algorithm in Gromacs is based on adjusting the size of the spatial domains decomposed onto MPI ranks. After benchmarking the enhancements supported by CRESTA, it became clear that the existing dynamic load balancing algorithm was not able to perform well enough on important scientific problems.

As an example, consider simulations of a protein embedded in a lipid membrane solvated in water, which are especially difficult in this sense. The distribution of the interactions is heterogenous, water is mostly electrostatic with some van der Waals required, lipid is normally only van der Waals and bonds but on the other hand the protein itself needs all three. Other kinds of simulations can have even bigger problems. The obvious course of action, as implemented in Gromacs 4.6, was to assign multiple cores to an MPI rank using OpenMP and hope that sufficiently large spatial domains can be used in order to alleviate the problem in practice. However, our hybrid MPI/OpenMP implementation was slower on CPU-only machines than a pure MPI at low-to-moderate scale and was useful only at the strong-scaling limit. This suggested the following steps:

- Look for improved implementations for distributing the workload within the cores of MPI ranks (to be discussed below).
- Attempt a redistribution of work not based on spatial locality.

In Gromacs 4.6 and 5, the spatial domains (i.e., the MPI ranks) which primarily handle water molecules have no bonded-interaction work to do, due to the water models being predominantly rigid. Cores of such ranks may lie completely idle. This is particularly a problem if accelerators are used and if the dynamic load balancing has assigned them an abnormally large spatial domain, although partial non-offload of short-range work would alleviate the problem somewhat. The domains that are mostly protein or lipid have a minimum size dictated by the implementation details of the non-bonded interactions (i.e., the short-ranged cut-off), so there is a minimum amount of bonded work that can only be performed on that rank. Load balancing problems arising from waiting for that work to complete can then limit the overall throughput.

Work is underway to redistribute bonded work evenly across a subset of MPI ranks, using non-blocking communication overlapped with short-range or PME work. This will improve performance through better load balancing, even if it does not directly improve strong scaling. Taking full advantage of such possibilities may require a better task-parallel implementation, as discussed below.

4.3.2 Task-based parallelism

Extensive preparations for full-scale conversion to task parallelism are in progress.

Conversion of the code base to compile as C++ is underway; now most of the ~1 million lines of performance-sensitive code compiles as C++. The conversion has exposed numerous minor bugs, and our continuous integration machinery runs valgrind, cppcheck, clang's static analyzer, Thread Sanitizer and Address Sanitizer, and also runs on five major compilers and five major operating systems in a bid to ensure highly portable, correct code.

Code transformations to express the bonded interaction kernels as tasks and the PME implementation in a task-parallel pipelined manner are underway. These transformations will make available some computation that can overlap in a task-parallel implementation once MPI 3 non-blocking collectives are deployed in PME.

These tasks can then be implemented and scheduled using a tasking framework, which will finally allow performance measurements of the suitability of TBB (or perhaps OpenMP or others).

Once these lessons are learned, the best path for expansion of the tasking scheme to other parts of the MD loop (update, constraints, short-ranged interactions) will become more clear.

4.3.3 Efficient large-scale I/O

When the user schedules periodic output of simulation information, the data can be packaged into a task and executed at low priority. For example, many cores currently lie idle during the hard-to-parallelize update, constraint and neighbour-search phases. This requires extra memory, but this is not a problem for MD simulations. It remains to be seen whether

- low-priority preprocessing tasks, followed by non-blocking transfer of output data to a master MPI rank, followed by asynchronous serial file I/O, or
- non-blocking MPI transfer of raw output data to a master MPI rank, followed by low-priority preprocessing tasks, followed by asynchronous serial file I/O, or
- low-priority preprocessing tasks, followed by non-blocking parallel I/O

is fastest in practice. The work to implement such a scheme is most efficiently done when the tasking scheme is in place (at which point, relaxing the serialization point might become truly necessary!)

4.3.4 Improved domain-decomposition halo exchange

The spatial domains that are the primary decomposition of work to MPI ranks in Gromacs need to exchange information on positions and forces of atoms of interest to neighbouring domains. This has to happen at each MD step. The implementation in Gromacs 4.0 used pulses along the three spatial grid dimensions to up to two successive neighbouring ranks [5]. This has been re-written to do a more classical halo exchange that communicates directly with the (up to) 7 neighbours in the “eighth-shell” decomposition used [4]. The new implementation directly anticipates improvements in intra-rank performance expected from task-parallelism, and prepares for FMM support and completion of more features of the Verlet cut-off scheme introduced in Gromacs 4.6 [8].

4.3.5 Multi-grid solvers for efficient PME electrostatics

Exascale performance in MD on scientific problems of interest will require the use of implementations for modelling long-ranged interactions that scale linearly with the number of cores. For instance, current implementations of PME require all-to-all communication patterns that will not scale well enough. The fast multipole method (FMM) has long been identified as a strong candidate to replace PME, because the performance of its implementation should scale linearly both in the number of particles and the number of cores. Until recently, the lack of available high-quality FMM implementations sufficiently general to be adapted for use in MD has impeded the progress. Two implementations of the FMM in Gromacs are now underway (only one is supported by CRESTA). These changes require significant adaptation of the existing Gromacs machinery for computing short-ranged interactions within a sphere to perform the components of the point-to-point component of the FMM in a (typically) non-spherical region.

4.4 Application performance and scalability

4.4.1 Efficient strong scalability, model problem

Trends in Gromacs strong scaling over the liftetime of CRESTA on an ion-channel benchmark case can be seen in Figure 4.1. The introduction of the Verlet scheme and the associated AVX support account for a large part of the improvements in performance in Gromacs 4.6 [8]. The availability of OpenMP in Gromacs 4.6 permits that scaling to extend to higher core counts, effectively relaxing the geometric constraints on the domain decomposition. Gromacs 5.0 adds support for the AVX2 instruction set, which works very well, though at higher core counts load imbalance and perhaps network configuration dominate the performance.

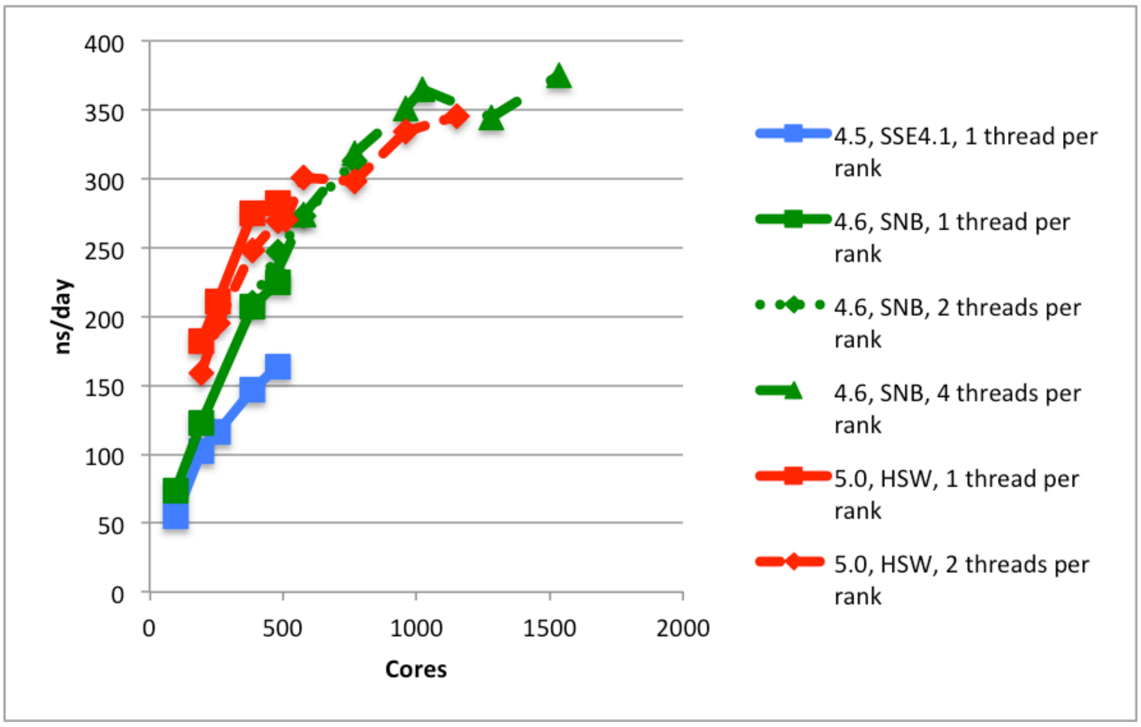


Figure 4.1 Scaling performance of Gromacs 4.5, 4.6 and 5.0 on Sandy Bridge (SNB) and Haswell (HSW) x86 platforms. Measurements with Gromacs 4.5 and 4.6 were done on triolith (8-core 2.2GHz Sandy Bridge nodes); measurements with Gromacs 5 were done on a machine with 2 16-core 2.3GHz Haswell processors per node. The simulation system was a 150K atom ion channel, using PME and 2fs time steps.

Strong scaling of Gromacs on BlueGene/Q also performs well on a range of benchmark cases (found in the CRESTA benchmark suite), as seen in Figure 4.2. In the maximally favourable case of the LJPME water simulation, where there is no external network noise, no per-rank load imbalance and a communication network which is relatively strong compared to the computing power of the processor, Gromacs 5.0 can scale down to 32 atoms/core. Higher throughput and lower scaling is available on commodity x86 hardware, however (see Figure 4.1).

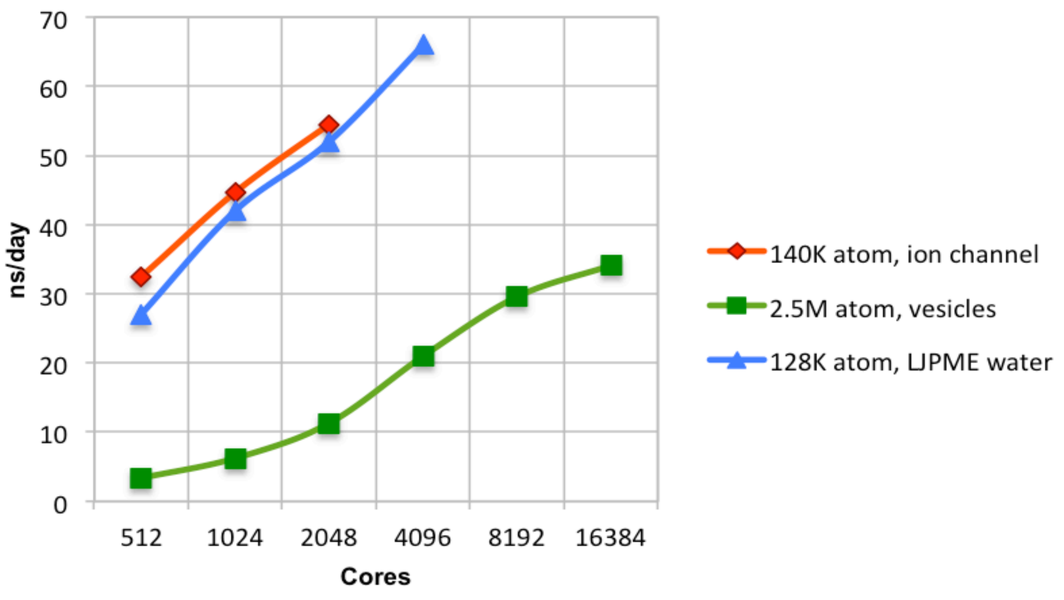


Figure 4.2 Scaling performance of Gromacs 5 on BlueGene/Q on three model simulation systems

Gromacs also has a first-class CUDA port, whose strong scaling performance is excellent. In Figure 4.3, we see the previously-reported scaling of the PME model in Gromacs 4.6 on GPUs on a range of similarly-sized model systems [2].

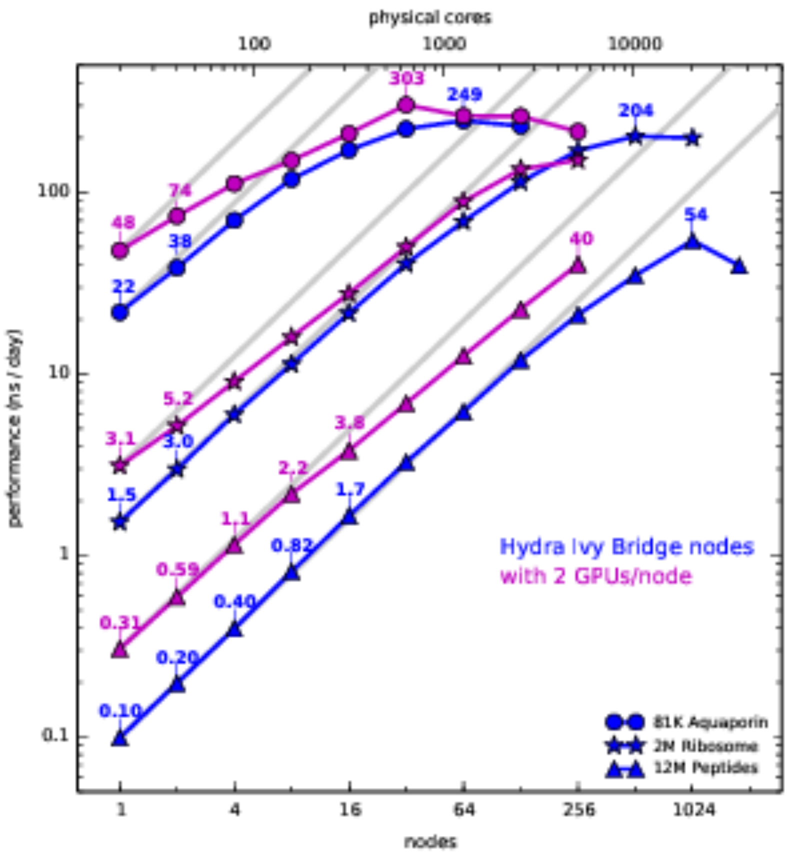


Figure 4.3 Strong scaling performance of Gromacs 4.6 on nodes with Ivy Bridge CPUs and two K20 GPUs.

4.4.2 Efficient weak scalability, model problem

Improvements in weak scaling capability is not a primary target for Gromacs, because the physical size and model resolution of scientific problems is essentially fixed. Nonetheless, for model physics with linear scaling, and apart from the known issue with writing trajectory output, weak scaling has been an essentially solved problem since Gromacs 4.5 [2].

Even for a non-linear scaling algorithm, such as the Lennard-Jones PME implementation in Gromacs 5, useful weak scaling can be seen in model problems such as the water simulation shown in Figure 4.4.

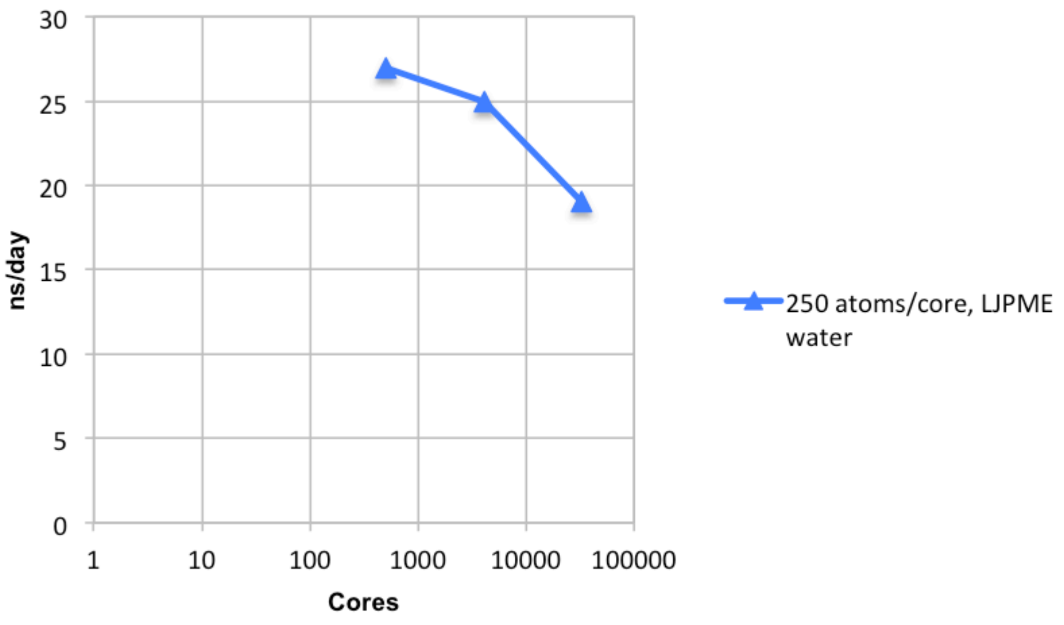


Figure 4.4 Gromacs weak scaling performance on BlueGene/Q on a non-linear-scaling physics model (combined Lennard-Jones+Electrostatic PME on a large box of water)

4.4.3 Efficient strong scalability, exemplar scientific simulation

Efficient strong scalability of Gromacs on an exemplar scientific simulation was demonstrated in CRESTA Deliverable 6.4 (see Table 4.6 and Figure 4.3) [10].

4.4.4 Efficient weak scalability, exemplar scientific simulation

Efficient weak scalability of Gromacs on an exemplar scientific simulation can be seen in CRESTA Deliverable 6.4 (see Table 4.6 and Figure 4.3) [10].

4.4.5 Summary of the performance improvements achieved

The two major reasons behind the performance improvements in molecular dynamics simulations in Gromacs achieved during CRESTA are:

- Rewrite of the short-ranged kernels and supporting neighbour-search code and the associated data structures, in order to directly and efficiently target the hardware characteristics of both CPU and accelerator cores [9].
- Addition of the support for hybrid MPI/OpenMP parallelism.

This has exposed further critical problems with load balance and intra-rank task scheduling, and exacerbated the known problems with communication load of both short- and long-ranged models. Work is underway to alleviate these.

The associated Copenicus software [4] makes it easy to deploy ensemble-style scaling of Gromacs on peta-to-exascale resources, and forms a key part of our exascale MD strategy. Since in exemplar scientific problems weak scaling is rarely of interest, the strong scaling will start to approach limits of available parallelism for fixed-size problems.

Also during the lifetime of the CRESTA project, ports to C++98 and the CMake build system were undertaken and an elaborate continuous integration testing infrastructure was built. A community code-review policy was introduced and works very well. Continued improvements in Gromacs performance would not be possible without such an infrastructure [3].

4.5 References

[2] Pronk S, Pall S, Schulz R, Larsson P, Bjelkmar P, Apostolov R, Shirts MR, Smith JC, Kasson PM, van der Spoel D, Hess B, Lindahl E, GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. Bioinformatics 29(7), 845-54 (2013).

[3] Pall S, Abraham MJ, Kutzner C, Hess B, Lindahl E, Tackling exascale software challenges in molecular dynamics simulations with GROMACS, Exascale Applications and Software conference EASC14, in press (2014).

[4] Pronk et al., “Copernicus: a new paradigm for parallel adaptive molecular dynamics”, SC11 High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for. IEEE, 2011. http://www.copernicus-computing.org.

[5] Bowers, K.J., Dror, R.O., Shaw, D.E.: Overview of neutral territory methods for the parallel evaluation of pairwise particle interactions. Journal of Physics: Conference Series 16(1), 300 (2005), http://stacks.iop.org/1742-6596/16/i=1/a=041.

[6] Hess, B., Kutzner, C., van der Spoel, D., Lindahl, E.: GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. J. Chem. Theory Comput. 4(3), 435–447 (2008)

[7] Spångberg D., Larsson D.S.D., van der Spoel D.: Trajectory NG: portable, compressed, general molecular dynamics trajectories. J. Mol. Mod. 17(10), 2669-2685 (2011) http://dx.doi.org/10.1007/s00894-010-0948-5.

[8] Lundborg M., Apostolov R., Spangberg D., Gardenas A., van der Spoel D., Lindahl E.: An efficient and extensible format, library, and API for binary trajectory data from molecular simulations. J. Comput. Chem. 35(3), 260-9 (2014) http://dx.doi.org/10.1002/jcc.23495.

[9] Pall, S., Hess, B.: A flexible algorithm for calculating pair interactions on SIMD architectures. Computer Physics Communications 184(12), 2641–2650 (2013), http://www.sciencedirect.com/science/article/pii/S0010465513001975.

[10] Exemplar scientific simulations, CRESTA Deliverable D6.4.

5 HemeLB

HemeLB is a tool for fluid flows in complex sparse geometries. Its main focus is simulating blood flow in parts of the cerebral arterial network. HemeLB employs an implementation of the lattice Boltzmann (LB) algorithm which, due to its locality, is intrinsically easy to parallelize. HemeLB uses MPI for communication and has been shown to have good scalability up to over 32k CPU cores.

5.1 Summary of the previous roadmaps

Task	Scheduled date	Status
Initial Roadmap		
Visualisation and Steering	M36	Completed
Pre-processing	M36	Completed
Introspection	M36	Completed
Roadmap update 1		
Single core performance	M20	Completed
Domain decomposition	M24	Completed and extended (see 3.3)
Hybrid parallelism	M30	In progress
Steerable parameter extraction	M30	Cancelled after partial result.
Visualisation	M36	Completed
Introspection	M36	Completed

Table 5.1 Summary of the previous roadmaps for HemeLB

Visualisation and Steering

To enable in situ visualisation and steering of HemeLB at the exascale, using visualisation libraries from WP5 partners.

Pre-processing

To enhance HemeLB's domain decomposition such that it is viable at exascale.

Introspection

Exascale applications will need to be able to monitor their own execution to be able to report and optimise their performance and the environment.

Single core performance

Based on our benchmarking and comparison to other Lattice-Boltzmann codes, we believe that there is scope to increase the single-core performance of HemeLB significantly. This will be undertaken with a fairly conventional profile-optimize cycle. We will be particularly interested in exploring the effect of changing the data layout to improve memory behaviour. This last piece of work will be undertaken in conjunction with the hybridisation task below.

Domain decomposition

Based on recent measurements, we see that some processes end up with a very large number of neighbours (~100) compared to the average (~25). These processes cause a load imbalance that is the primary cause of the sub-linear scaling we see at 32k cores. We are working with partners in WP5 to trial the PPStee domain decomposition library in order to improve this.

Hybrid parallelism

Based on the report by Alan Gray which was the main output of our co-design work, we will not pursue OpenACC until the software is more mature. However he has shown that OpenMP is much more feasible and we will work on this further. The effect of different memory layouts will be explored in detail here.

Steerable parameter extraction

In HemeLB we have implemented a property extraction framework that allows the user to define regions of interest for output, as well as which fields to output and at what frequency. Currently this must be specified at simulation start. We propose to make this part of the code steerable at run time, in order to allow the user to quickly home in on interesting features which can be recorded for more detailed off-line analysis.

Visualization

We will continue to work with WP5 partners to explore how to couple their visualisation software with HemeLB.

Introspection

We have implemented key introspection abilities ourselves which are sufficient for our needs. We will continue to monitor developments within this arena, but our judgment is that applications will need to delegate the responsibility of monitoring and acting to runtime systems, since the complexity and variability of future systems will likely be too large for an application to find a generic solution. We have therefore paused this activity until suitable technology is available and there is a compelling need for it.

5.2 Achievements towards remaining tasks

Task	Achievement
Hybrid parallelism	In progress. We have performed a preliminary study with OpenMP as well as OpenACC. The end result of the study was that in particular an OpenACC port is challenging for a C++ code like HemeLB. Based on this outcome, we are now preparing a major port to many-core architectures. We are also pursuing hybrid parallelism using ensemble techniques (see 3.3), and have investigated the use of non-blocking collectives in HemeLB.
Steerable parameter extraction	The HemeLB steering client has previously been recommisioned and benchmarked for performance. For sustainability reasons we have hooked up HemeLB with the DLR steering client and suspended any further development on the HemeLB steering client.

Table 5.2 Achievements towards remaining tasks from the previous roadmaps for HemeLB

5.3 Roadmap to exascale

Task	Estimated effort	Status
Automated ensemble simulation approach	6 PM (4 invested)	Ongoing, preliminary results available.
Stand-alone domain decomposition tool	4 PM (2 invested)	Ongoing, preliminary results available.
Port to many-core architectures	18 PM	In planning.

Table 5.3 Roadmap to exascale for HemeLB

5.3.1 Automated ensemble simulation approach

Arteries are subject to a wide range of flow regimes through a patient's life. Thus comprehensive analysis of flow dynamics within a cerebral artery, a prime requirement for assessing risks of aneurysm formation and rupture, cannot be performed using a single simulation instance. In this task we construct an automated approach to create initial conditions, and instantiate an ensemble of HemeLB simulations, with different instances being subject to flow inputs with differing heart rate and blood pressure values. This task is important for the exascale for two reasons: (I) Exascale machines will be highly expensive to use, and by performing a comprehensive flow analysis of cerebral arteries we can more convincingly justify the use of such infrastructures. (II) Through the incorporation of a range of likely and realistic flow regimes, the computational requirement of HemeLB will increase by at least one order of magnitude. We have started on this task during CRESTA, and plan to finalize it in subsequent projects.

5.3.2 Stand-alone domain decomposition tool

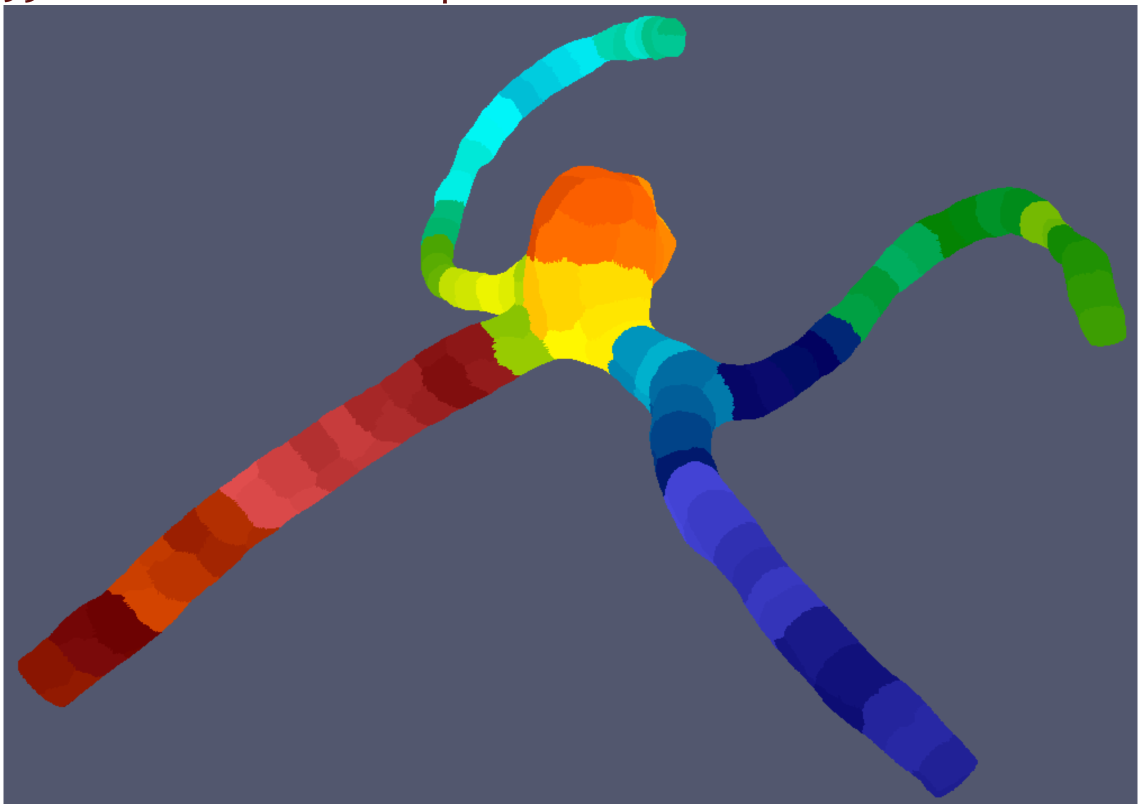


Figure 5.1 Example visualization of a domain decomposition on a 4.6 million lattice site aneurysm geometry, partitioned into 128 fragments. We converted the data with an early version of protopart and relied on PPStee directly (no HemeLB run required).

The lack of a balanced domain decomposition is a major constraint for simulation performance in solvers relying on sparse geometries [1]. In addition, we observed in HemeLB that the domain decomposition step fails to scale on larger core counts, and that we are required to repeat identical domain decomposition steps when running multiple instances in an ensemble. In this task we construct a stand-alone domain decomposition tool (with protopart as the working name), and aim to perform the domain decomposition step outside of HemeLB, improving the scalability of the main code. In addition, we can experiment with different decomposition approaches without running instances of HemeLB, accelerating our search for optimal partitioning methods. We have developed a first version of *protopart* within CRESTA. In addition, we have a funded eCSE project, which will allow us to integrate the library with HemeLB.

5.3.3 Port to many-core architectures

Porting HemeLB to many-core architectures will be a major undertaking, but given the recent trends (e.g., the emergence of the CORAL architecture [2]) it is likely that such a port will be essential to complete our preparation to the exascale. We intend to perform this comprehensive port as part of upcoming projects.

5.4 Application performance and scalability

In this subsection, we consider the performance and scalability of HemeLB before and after the CRESTA project.

5.4.1 Efficient strong scalability, model problem

As model problems we use two different cases, representing two very different types of problems. The Huge cylinder test case is simply a large cylinder with 230M lattice sites with an aspect ratio (length to height) of about 2:1.

Our second test case resembles an actual scientific use case better than the first one. It is called Arterial bifurcation and it represents a forked artery consisting of 25M lattice sites. It has a fluid fraction of around 11%, i.e., it is sparse in terms of lattice sites. The results for both test problems, described in lattice site updates per second, are given in Table 5.4.

With the Huge cylinder test case we obtain a perfect parallel speedup in terms of site updates per second when doubling the number of cores. For the first two core counts, i.e., 12,288 and 24,576, the number of lattice sites per code is about 18,000 or 9000, respectively. Further doubling of the input data reduces the number of lattice sites per core to about 4500 and gives a parallel speedup of 1.49. The drop in parallel efficiency is caused by the rather small local problem size per core as the computation versus communication ratio becomes too small to achieve perfect scaling.

Due to its sparsity, it is much harder to achieve a good parallel speedup for the second test case. In Table 5.4 we have given results only for the largest test core count tested, where we obtained 43 billion site updates per second with only about 2000 lattice sites per core. Due to the sparsity of the domain, the second test case resembles the exemplar scientific simulation more than the first one. Thus, when the core counts are similar, we can expect the performance of the second test case to give an upper bound to the lattice point updates per second for the exemplar scientific test case.

Domain type	Core count	Site updates per second
Huge cylinder (~230 million lattice sites)	12,288	51 billion
	24,576	103 billion
	49,152	153 billion
Arterial bifurcation, 20 micrometer voxel size (~25 million lattice sites)	12,288	43 billion

Table 5.4 HemeLB -performance for a model problem

5.4.2 Efficient strong scalability, exemplar scientific simulation

As part of Deliverable “D6.4 Exemplar scientific simulations” [7], we have performed several runs with a Circle of Willis geometry, consisting of 73 million lattice sites. We note that the Circle of Willis geometry is very sparse in terms of lattice sites. Such sparsity has two main detrimental effects on performance: firstly, it reduces the beneficial caching effects on a CPU. Secondly, the uneven sparsity of the domain leads to more load imbalance in the domain decomposition step. Any such load imbalance is in turn made worse by the lower computation versus communication ratio of a sparse domain.

Simulation	Description of run			
	Nodes	Total cores	Wall time	Site updates per second
Circle of Willis Scaling test, 10k time steps, no I/O	32	768	235.0	3.1 billion

Circle of Willis Scaling test, 10k time steps, no I/O	64	1536	132.0	5.5 billion
Circle of Willis Scaling test,10k time steps, no I/O	128	3072	68.7	10.6 billion
Circle of Willis Scaling test,10k time steps, no I/O	256	6144	37.1	19.7 billion
Circle of Willis Scaling test,10k time steps, no I/O	512	12288	23.2	31.5 billion
Circle of Willis Scaling test,10k time steps, no I/O	1024	24576	25.3	28.9 billion
Circle of Willis Production test, 800k time steps, I/O every 10k time steps	1024	24576	2270.0	25.7 billion

Table 5.5 HemeLB - exemplar simulation runs on ARCHER

5.4.3 Summary of the performance improvements achieved

As testified by the task list in their previous sections, and their current status, we have performed a huge range of optimization activities within the CRESTA project. Here we summarize the improvements achieved in HemeLB, proceeding first with an overview of the performance improvement and next with an overview of the other benefits obtained during the CRESTA project.

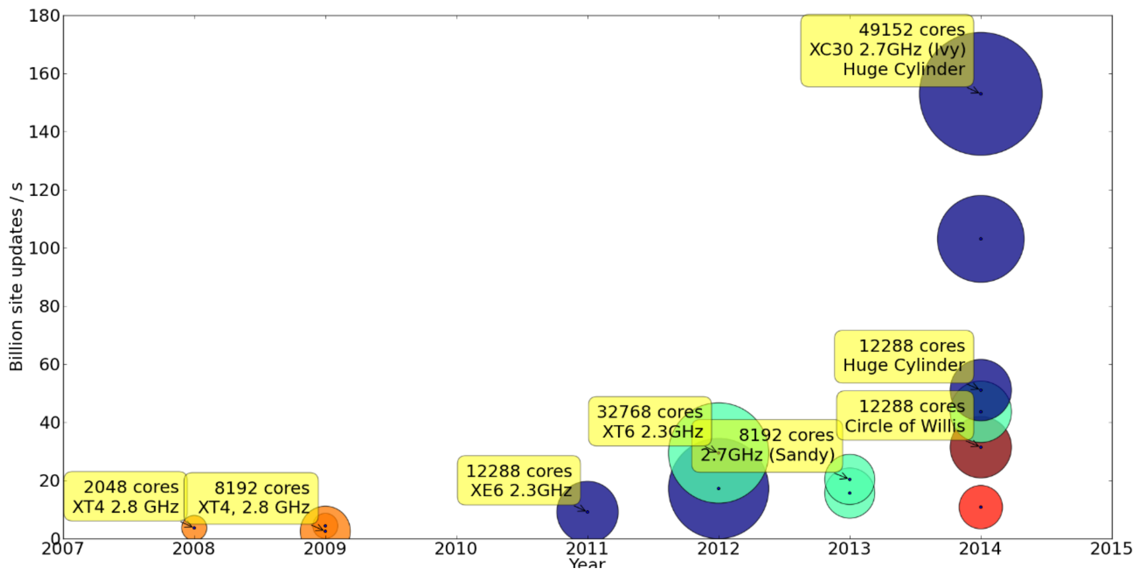


Figure 5.2 Obtained maximum performance achieved with HemeLB between 2007 and 2014. All improvements after 2011 were achieved during the CRESTA project. The sparsity of the data sets is roughly indicated by the colour of the circle (very sparse is red, non-sparse cylinder data sets are blue), and the core count used by the size of the circle

Raw Performance Power

We show the development of the maximum achieved performance of HemeLB over the years in Figure 5.2, as published in papers and technical reports. At the start of the CRESTA project, in 2011, we were able to obtain a performance of 9.1 billion site updates per second using 12,288 cores. As a result of our collaborations in CRESTA, we now have been able to obtain a performance of 153 billion site updates per second using 49,152 cores in 2014. This constitutes a performance improvement of a factor 16.8, whereas Moore's law would predict a performance improvement of approximately a factor 2.8 over that period.

Although the maximum obtained number of site updates per second is a good indicator of how HemeLB is able to make more efficient use of high-end computing resources, it is only one of two important performance aspects for scientist users in the field. This is because in lattice-Boltzmann simulations the number of time steps required to reach convergence tends to scale with the size of the system modelled. As such, the number of time steps we can take per second of wall-clock time in HemeLB becomes increasingly important when we simulate larger problem sizes. In Figure 5.3 we present this measure for all the runs that we previously presented in Figure 5.2. We find that our efforts within CRESTA have allowed us to simulate larger geometries than ever, and that we have managed to increase the rate of simulation to almost 2000 time steps per second. We believe these benefits are mainly results from our single-core optimizations, our improvements in domain decomposition, and the advent of newer and faster architectures such as Intel Ivy Bridge.

For very large data sets (e.g., the huge cylinder), however, we do observe a lower speed of about 600 time steps per second. Indeed, one of the major future challenges for HemeLB will be to improve this measure for large problems, allowing us to reach convergence for these simulations within reasonable time spans.

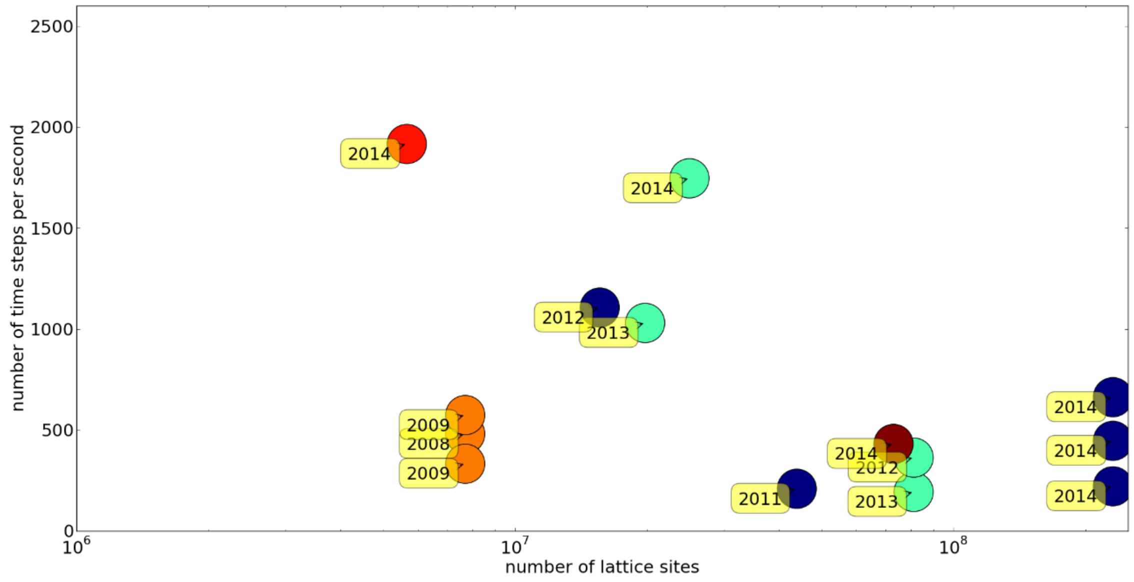


Figure 5.3 Obtained maximum number of time steps per second achieved, as a function of the problem size in the simulation (measured in number of lattice sites).

Other benefits to HemeLB

We have obtained a number of other benefits for HemeLB over the course of CRESTA. These are listed in what follows.

More robust domain decomposition routines

In addition to obtaining a better load balance, we are now also able to do domain decompositions reliably on larger geometries, due to enhancements in the configurations and use of the underlying partioning libraries. This has allowed us, for example, to reliably run simulations of a 73M lattice site Circle of Willis geometry.

Improved visualization and steering

We have connected HemeLB to the DLR Steering and Visualization client, providing us with more robust and sustainable software infrastructure to perform *in-situ* visualization and steering of ongoing HemeLB simulations.

Improved inter-process intercommunication

Early in the CRESTA project we implemented the coalesced communication software pattern [3], allowing us to use sophisticated non-blocking techniques within HemeLB and systematically organize the different communication *concerns* in the code (e.g., communication required for simulation, visualization or monitoring). In the last year we have also implemented and tested the CRESTA non-blocking collectives in HemeLB, achieving comparable performance while simplifying the code base.

Improved diagnostics

We developed the Property Extraction Framework, which now allows us to conveniently extract selected macroscopic quantities from our simulations. This enhancement has made HemeLB considerably easier to use for scientific purpose, and allows us to work on pre-filtered data sets, reducing the I/O requirements of HemeLB simulations.

Improved code organization, usage and diagnostics

As part of our numerous enhancements to HemeLB, we have incorporated a wide range of unit tests and continuous integration tests. In addition, we have created a Python-based environment to provide shorthand commands for compiling, configuring and executing HemeLB simulations on remote resources. Last, but not least, we have hooked up HemeLB with essential diagnostic tools such as Allinea MAP, DDT and the MUST checker for MPI correctness (see D3.11 for a comprehensive description of those activities [8]).

More science

Although the direct pursuit of domain-specific research with HemeLB is not a direct goal in CRESTA, we argue that it does play an important role in proving that the work done in CRESTA has had major benefits for the domain-specific research activities with HemeLB, raising its profile in the scientific community. For example, using HemeLB, we have been able to publish new advances in high-profile domain journals such as J. R. Soc. Interface [4], Physics Review E [5] and Interface Focus [6].

5.5 Code comparison: HemeLB, JYU-LB, AboLB

5.5.1 Background

The lattice Boltzmann method has emerged as a potential simulation task that can scale to exaflops and beyond. In order to explore the level of performance that can be reached for this kind of simulation, two new codes, the CPU code JYU-LB and the GPU code AboLBM, were developed in CRESTA and compared against HemeLB for strong scaling performance. For our comparison three different simulation geometries where used. The first case was a square duct or empty box, essentially a sample with only fluid and no solid sites and with a volume of 512 cubed. The second was a porous media sample representing a sandstone sample with a size of 1024 cubed with a porosity of around 13%, obtained from R. Hilfer et al. at the Institute for Computational Physics at the university of Stuttgart [11]. The last sample is the Circle of Willis (CW) which is the main blood distribution system in the brain. It is a ring-like structure connecting the internal carotid arteries with the cerebral arteries via a set of communicating arteries. The geometry used was kindly provided by Prof Figueroa at KCL [10] and was discretized for simulation with a grid spacing of 3.3e-5 m.

5.5.2 Test Environment

For the CPU tests ARCHER at EPCC was used. ARCHER is a Cray XC30 machine consisting of 4920 compute nodes, each with two 12 core Intel Xeon E5-2697 v2 processors connected together with the Cray Aries interconnect for a total of 118 080 cores with a peak performance of 1.642 PFLOPs.

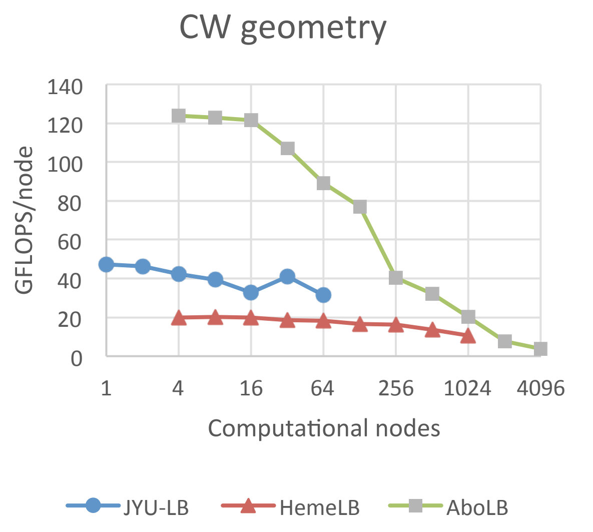
For our GPU tests we used Titan at Oak Ridge. Titan is a Cray XK7 machine with 18688 compute nodes, Each node consists of one AMD Opteron 6274 16 core CPU and one Nvidia Kepler K20x GPU giving a total peak performance of 17.590 PFLOPs.

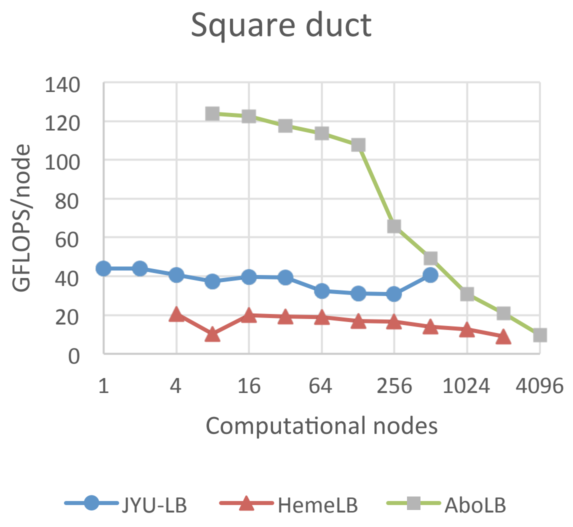
5.5.3 Codes

We compared two CPU codes and one GPU code with each other. The first CPU code is HemeLB, the inner workings of which has been presented earlier, in this case HemeLB was run using the D3Q19 stencil with half way bounce back boundary conditions For the relaxation model a newly implemented two relaxation time (TRT) model was used.

The second CPU code, referred to as JYU-LB, was developed at University of Jyväskylä. It uses the D3Q19 stencil and the TRT collision model. No-slip boundary condition is implemented with the common halfway bounce-back scheme. Time propagation is realized with the AA-pattern [9] algorithm that uses a fused implementation, where the relaxation and propagation steps are executed together for each lattice site. JYU-LB is parallelized with hybrid strategy using MPI communication over computation nodes and OpenMP inside the nodes. Subdomains designated to nodes therefore have close to optimal load balance over threads. However, JYU-LB only supports Cartesian decomposition (rectangular subdomains) which greatly simplifies the code, but deteriorates load balance between nodes for non-homogeneous geometries. Using an efficient differential evolution optimization algorithm we adjust the positions of the hyperplanes of the Cartesian decomposition attempting to equalize the number of fluid cells inside the subdomains.

The GPU code, referred to as AboLB, was developed at Åbo Akademi University. It is similar to the JYU-LB in that it is implemented using the same relaxation operator, boundary conditions as well as time propagation algorithm. Instead of using OpenMP to parallelize the computational part however it is offloaded to the GPU using CUDA. The code also implements asynchronous communication where the CPU will make progress on the communication while at the same time the GPU is doing the necessary computation for the current time step. AboLB also supports a more general load balancing scheme based on rectangular subdomains, and in these comparisons the load was distributed using a simple recursive bisection approach.





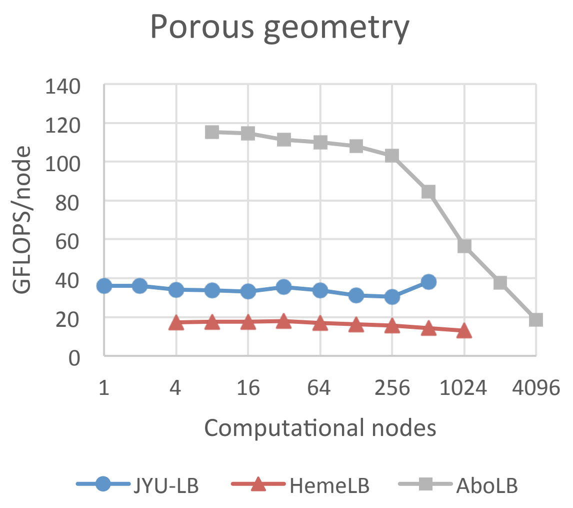


Figure 5.4 Results for the LB comparison between JYU-LB, AboLB and HemeLB.

5.5.4 Results

The three different samples were run using the three different solvers, measuring strong scaling from 1 to 4096 nodes using the same input data for all codes and dividing the computational domain into smaller parts as needed. The flops numbers are derived by measuring the number of site updates per second the codes achieve and then multiplying the result with the number of floating point operations each site update consists of on that architecture. For the CPU solver the number of floating point operations is estimated to be 219 while on the GPU the number of floating point operations was measured using a profiler to be 279 operations per site update. The strong scaling performance results for the three lattice Boltzmann solvers using the three geometries are given in the graphs, measured as GFLOPs/node. Ideally this number stays constant.

5.5.5 Strong Scalability: Conclusions

Both CPU based codes benefit from the faster interconnect offered by the Cray XC30 machine. Additionally the Aries interconnect on the XC30 machines is also a more advanced layout, a dragonfly topology with a fixed worst case number of hops, compared to the torus interconnect on Titan where the maximum number of hops between two specific nodes varies based on the status of the machine and the job size.

The JYU-LB shows excellent strong scalability for all geometry cases. It scales from one compute node to the maximum number of nodes tested without the per node performance dropping below half of the initial per node performance. The slowest combination of nodes only drops down to 0.7 of the initial performance for the square duct, 0.66 for the CW geometry and 0.84 for the porous geometry. For the square duct and porous geometry cases the code also achieves super linear scaling, providing the best performance per node at the largest node count the code was run on. The super linear scaling phenomena occurs when all the data associated with the current simulation fits into the L3 cache on the CPUs, in this case the CPUs have 30MB of L3 cache each for a total of 60MB.

HemeLB also shows great scalability for the test geometries, only dropping below half of the initial per node performance for the square duct case when scaling to 2048 compute nodes. The slowest per node performance compared to the initial performance for the CW case was 0.74, and 0.53 of the initial performance for the porous case.

AboLB, the GPU solver appears to have some scalability issues. For all geometries the performance quickly drops to less than half of the initial per node performance: for the square duct this occurs when scaling to 512 nodes, for the CW case when scaling to only 256 nodes and in the porous case AboLB performed the best but the performance per node still fell slightly below half the initial per node performance when scaling up to 1024 nodes. The code starts scaling poorly when the simulation becomes communication bound, for the CW case some nodes start missing their communication deadlines at 32 nodes already, at 256 nodes almost all nodes miss all their communication deadlines. The Square duct case also start missing all its communication deadlines when scaling to 256 nodes while the porous media case fairs slightly better due to the lower ratio of data that needs to be communicated compared to the total computation per node. The issues are partly due to the network on Titan, but another factor to consider is also the fact that the GPU is an added component of the system, meaning that any communication needs to make an additional hop over the PCI-e bus. Although the added latency from this extra hop is significantly smaller than the latency of the node to node communication it still does affect the performance somewhat.

5.5.6 Floating point performance: conclusions

Peak numerical performance for the CPU based codes were achieved at the maximum number of nodes they were executed on. For HemeLB peak performance was 18371.05 GFLOP for the duct case on 2048 nodes, 13158.33 GFLOPS for the porous media case on 1024 nodes and 10807.85 GFLOPS for the CW case on 1024 nodes.

JYU-LB reached a peak performance of 20876.2 GFLOPS for the duct case on 512 nodes, 19395.3 GFLOPS for the porous media case on 512 nodes and 2005.58 GFLOPS on 64 nodes for the CW case. For the AboLB peak performance was reached on 2048 nodes for all cases and not the maximum 4096 nodes the samples were scaled to. For the GPU the peak performance was 42896.6 GFLOPS for the duct case, 77228.28 GFLOPS for the porous media case and 15595.9 GFLOPS for the CW case.

Comparing all solvers at 64 compute nodes and all different geometry samples gives a more even comparison. It is clear that from a total floating point performance standpoint the AboLB GPU based code is the fastest, all of the additional speed however is not just due to the fact that the GPUs offer a better theoretical floating point performance but also due to the massive additional bandwidth available on them. Most of the difference between the CPU solvers can be attributed to the AA algorithm used by JYU-LB.

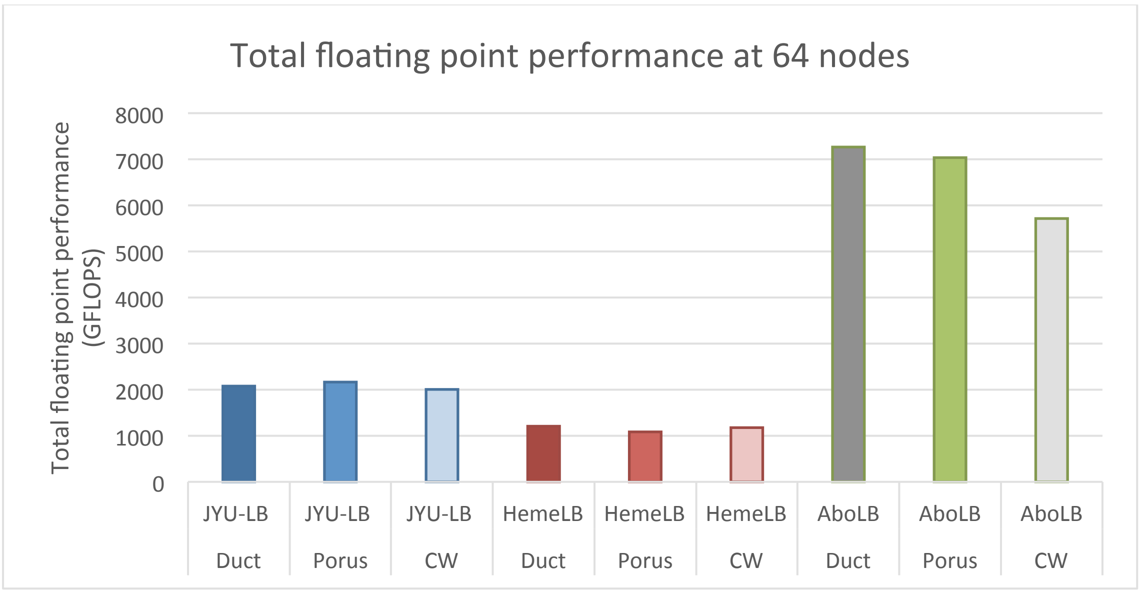


Figure 5.5 Total floating point performance for the three LB codes: JYU-LB, AboLB and HemeLB.

Both JYU-LB and AboLB have also completed large scale runs using a higher resolution version of the porous media sample, 16384 cubed instead of 1024 with the same porosity. JYU-LB was run on Archer using 2880 nodes (96% of the Phase 1 configuration) and was able to reach a performance of 0.078 PFLOPS. AboLB has been run on 16384 nodes of Titan using half of the high resolution porous media sample, only half the sample was used due to memory constraints with less than 6GB available per GPU. Peak floating point performance for this

5.6 References

[1] "Weighted decomposition in high-performance lattice-Boltzmann simulations: are some lattice sites more equal than others?", D. Groen, D. Abou Chacra, R. W. Nash, J. Jaros, M. O. Bernabeu, P. V. Coveney, EASC 2014 Proceedings (in press), arXiv preprint arXiv:1410.4713

[2] http://www.hpcwire.com/2014/11/14/coral-signals-new-dawn-exascale-ambitions/

[3] "Coalesced communication: a design pattern for complex parallel scientific software", H.B. Carver, D. Groen, J. Hetherington, R.W. Nash, M.O. Bernabeu,

P.V. Coveney, submitted to *Advances in Engineering* Software, arXiv preprint arXiv:1210.4400, 2012.

[4] “Computer simulations reveal complex distribution of haemodynamic forces in a mouse retina model of angiogenesis”, M.O. Bernabeu, C.A. Franco, M. Jones, J.H. Nielsen, T. Krüger, R.W. Nash, D. Groen, J. Hetherington, H. Gerhardt, P.V. Coveney, J. R. Soc. Interface (in press), arXiv preprint arXiv:1311.1640, 2013.

[5] “Choice of boundary condition for lattice-Boltzmann simulation of moderate Reynolds number flow in complex domains”, R.W. Nash, H.B. Carver, M.O. Bernabeu, J. Hetherington, D. Groen, T. Krüger, P.V. Coveney, Physics Review E 89, 023033, 2014.

[6] “Impact of blood rheology on wall shear stress in a model of the middle cerebral artery”, M.O. Bernabeu, R.W. Nash, D. Groen, H.B. Carver, J. Hetherington, T. Krüger, P.V. Coveney, Interface focus 3 (2), 20120094, 2013.

[7] Exemplar scientific simulations, CRESTA Deliverable D6.4.

[8] Experiences with benchmarks and co-design applications, CRESTA Deliverable D3.11

[9] Bailey, P., Myre, J., Walsh, S., Lilja, D., & Saar, M. (2009). Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors. *International Conference on Parallel Processing, 2009. ICPP '09.*

[10] Coogan, J., Humphrey, J., & Figueroa, C. (2013). Computational simulations of hemodynamic changes within thoracic, coronary, and cerebral arteries following early wall remodeling in response to distal aortic coarctation. *Biomechanics and Modeling in Mechanobiology* (12(1)), 79-93.

[11] Hilfer, R., & Zauner, T. (2011). High-precision synthetic computed tomography of reconstructed porous media. *Phys. Rev. E , 84* (6), 062301.

6 IFS

The Integrated Forecasting System (IFS) is the production numerical weather forecast application at ECMWF. IFS comprises several component suites, namely, a 10-day high-resolution forecast model, a four-dimension variational analysis (4D-Var) for producing the initial conditions for the forecast, an ensemble prediction system and an ensemble data assimilation system.
The use of ensemble methods are well matched to today's HPC systems, as each ensemble application (model or data assimilation) is independent and can be sized in resolution and by the number of ensemble members to fill any supercomputer. However, these ensemble applications are only part of the IFS production suite and the high resolution forecast model (referred to as 'IFS model' from now on) and 4D-Var analysis applications are equally important in providing forecasts to ECMWF member states of up to 10 to 15 days ahead.
For the CRESTA project it has been decided to focus on the IFS model to understand its present limitations and to explore approaches to get it to scale well on future exascale systems.

6.1 Summary of the previous roadmaps

Task	Scheduled date	Status
Coarray kernel	4Q2011-1Q2012	Completed
IFS CY37R3 port	1Q2012	Completed
Legendre transform coarray optimization	2Q-3Q2012	Completed
Semi-Lagrangian coarray optimization	4Q2012-2Q2013	Completed
Optimization of Fourier latitude load-balancing heuristic	2013	Completed
Development of a future solver for IFS	2014	CRESTA contribution completed (see below)
Fourier transform coarray optimization.	3Q2012-4Q2012	Completed
IFS CY38R2 port	1Q2013	Completed
Radiation in parallel scheme (added)	1Q2013-1Q2014	Completed
Investigate GPU use in IFS	2Q2013-4Q2013	Completed
Investigate graph based (DAG) parallelization	2H2013-2014	Completed

Table 6.1 Summary of the previous roadmaps for IFS

Coarray kernel

Develop kernel to investigate overlapping computation and communication using Fortran2008 coarrays in an OpenMP parallel region.

IFS CY37R3 port

Port of IFS model (CY37R3) to HECToR and analysis of performance for model resolutions up to T2047 (10km grid).

Legendre transform coarray optimization

Optimization of the IFS transform library to overlap the computation of the Legendre transforms with the associated communications (TRMTOL/TRLTOM).

Semi-Lagrangian coarray optimization

Developments to the IFS semi-Lagrangian scheme to use Fortran2008 coarrays to improve scalability by removing the need to perform full halo wide communications. In addition, computations in the semi-Lagrangian scheme to determine the departure point and mid-point of the trajectory are overlapped with coarray transfers from neighbouring tasks.

Optimization of Fourier latitude load-balancing heuristic

Optimization of the heuristic used to statically load-balance the distribution of variable length latitudes in grid-space. An optimal distribution of latitudes is required to load-balance the cost of performing Fourier transforms as IFS transforms data from grid to Fourier space. Work on this task quickly showed that the best static load-balancing heuristic at scale was to load-balance the latitude data and ignore the FFT computation imbalance. To achieve the perfect data load-balance required a rewrite of the trans library routine sumplatb_mod.F90 which was also tested offline to beyond 1M cores (assuming 16 threads per task).

Development of a future solver (alternative dynamical core option) for IFS

Research into a dynamical core for extreme scaling of IFS and a potential replacement of the spectral method. This research is primarily being performed at ECMWF within the Numerical Aspects section and in particular with Piotr Smolarkiewicz who is a recipient of a European Research Council grant (project “PantaRhei”) in the Seventh Framework Programme (FP7/2012/ERC Grant agreement no. 320375). Recent developments in this section are presented in [1] and [2]. Within CRESTA, supporting research has focused on developing a flexible computational environment to provide extreme scalability with predominantly nearest neighbor communication. Here, spatial discretization employs bespoke unstructured meshes built about the vertices of the reduced Gaussian grid employed in IFS as shown in Figure 6.1. Such an arrangement allows using a domain decomposition identical to IFS and opens avenues to the future high fidelity comparisons with IFS’ solutions of primitive equations. Furthermore, it allows for model hybridization where selected elements can be directly exchanged between the new dynamical core option and IFS, without interpolation. The development operates on flexible dual meshes with an efficient parallel edge based data structure and a non-staggered arrangement of flow dependent variables. Tests have recently progressed to running selected climate benchmarks of global shallow-water flows. A good resource describing such benchmarks is contained in [4]. It should be noted that the overall development of an alternative dynamical core is estimated to take in the order of 10 person years. The CRESTA contribution to this effort is now complete in respect of the development of a supporting “Atlas” library providing the flexible framework discussed above. A pre-release version of the Atlas library has been uploaded to the CRESTA source repository.

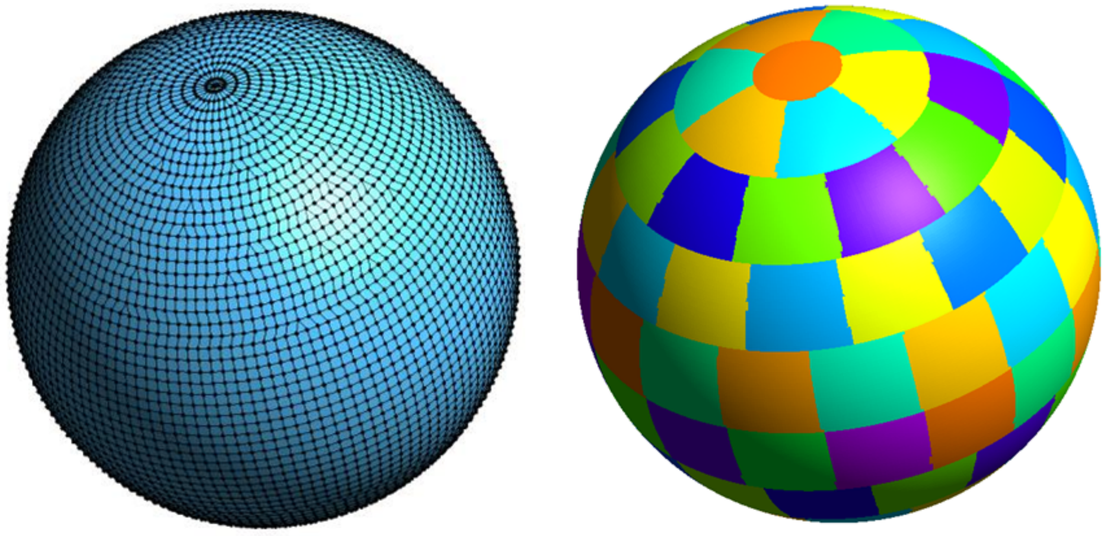


Figure 6.1 IFS T63 mesh (nodes are existing T63 grid) and existing EQ_REGIONS partitioning

Fourier transform coarray optimization

Optimization to the IFS transform library to overlap the computation of the Fourier transforms and Fourier space calculations with the associated communications (TRGTOL/TRLTOG). This was omitted from the D6.1.1 schedule.

IFS CY38R2 port

Port IFS model code version CY38R2 to HECToR. This code cycle became available in 4Q2012 and included support for the T_L3999 (5 km) model resolution and fast Legendre transform. This code version was packaged as a RAPS13 benchmark and included all the IFS Fortran2008 coarray optimizations implemented in the first year of the CRESTA project.

Run T_L3999 IFS model (5 km global model):

This subtask was completed on TITAN in 2Q2014 where a 5km IFS global model was run using a T_C1999 cubic grid with half the spectral resolution of the T_L3999 linear grid we had originally proposed, as reported in D6.4. It should be noted that T_C1999 and T_L3999 have exactly the same number of grid-points.

Assess coarray optimizations at T_L3999:

This subtask was completed in 2Q2014 as above and reported in D6.4.

Radiation in Parallel Scheme

A scheme whereby the IFS radiation computations were executed in parallel with the rest of the model was implemented and reported in [3]. This work showed that in principle it is possible to expose greater parallelism by such an approach, however, realizing a performance improvement requires computations that are run at every time-step and are well balanced with the rest of the model in terms of processor resources.

Investigate GPU use in IFS

Some initial experience of using GPUs was performed as part of the TITAN INCITE14 award to the CRESTA project. Originally we planned to explore intercepting the matrix-matrix multiplies (DGEMMs) that are called in the Legendre transforms and execute them on GPUs. What we actually did was to port the whole spectral transform scheme used in IFS involving Legendre transforms, Fourier transforms and data transpositions used in an IFS time-step, and cycle this for 100 time-steps. While this represented a very small fraction of the IFS source it did provide some useful experience with using OpenACC and the NVIDIA cuFFT library. Figure 6.2 shows a compute cost performance comparison for a Tc1999 spectral transform test running on 140 TITAN Nvidia K20X GPUs against 140 Cray XC-30 nodes each with 24 Intel Ivybridge cores. In Figure 6.2 the TITAN K20X GPU computation cost excludes the associated data transfers between host and GPU for each computation section. These host/GPU transfers are included in the K20X GPU MPI communication costs. Table 6.2 shows the actual costs shown in Figure 6.2, and includes a simple prediction for a XC-30 system (with a minimal number of XC-30 cores and using the Aries MPI interconnect) with a K20X GPU. The prediction indicates the full cost of the spectral transform test could be a little faster with a GPU approach, and about a factor of 2 reduction in power cost. This is based on data that a Cray XC30 24 core Ivybridge node uses 2.4 times the power of Nvidia K20X GPU.

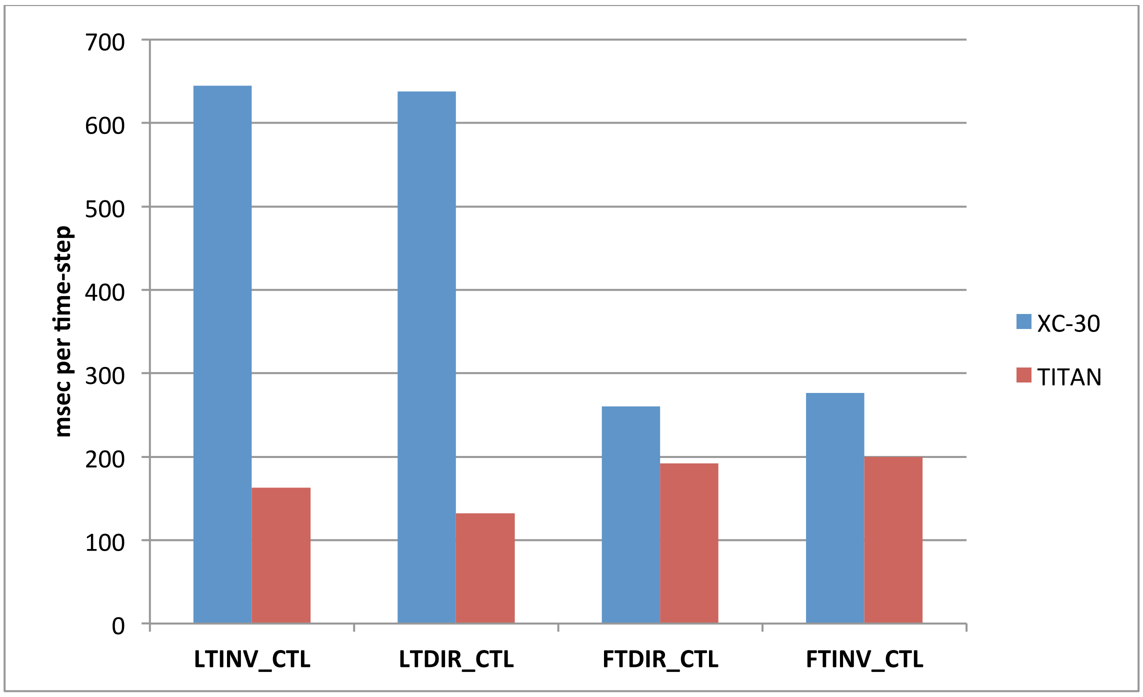


Figure 6.2 T_C1999 5 km model spectral transform test compute cost (140 full nodes, 800 fields)

T_c1999	XC-30	TITAN K20X GPU	XC-30 (Aries) +GPU Prediction
LTINV_CTL	645.2	162.8	162.8
LTDIR_CTL	638.0	132.0	132.0
FTDIR_CTL	260.2	192.4	192.4
FTINV_CTL	276.6	199.8	199.8
MTOL MPI	547.3	1564.9	547.3
LTOM MPI	222.8	1633.6	222.8
LTOG MPI	502.0	975.8	502.0
GTOL MPI	191.7	998.7	191.7
HOST2GPU	-	376.0	376.0
GPU2HOST	-	285.0	285.0
Total	3283.8	5860.0	2811.8

Table 6.2 Tc1999 5 km model spectral transform test performance, in milliseconds (140 full nodes, 800 fields)

Investigate graph based (DAG) parallelization

The use of graph based parallelization in IFS has been investigated. For this work a small (1100 line) kernel was coded with MPI/OpenMP parallelization, to simulate the computations and data dependencies in IFS physics and Fourier transforms. OmpSs tasking model [6] was used for the DAG parallelization and we had many good exchanges with Prof. Jesus Labarta at the Barcelona Supercomputer Centre. This work showed promising results for small numbers of threads per task. As the thread count was increased the MPI/OpenMP version was always faster than the MPI/OmpSs version. The slower OmpSs performance was identified to be a consequence of having to restrict MPI communications to one thread per OmpSs process as the available MPI implementation was not thread safe. Both two-sided and one-sided MPI communications were explored in this work.

6.2 Achievements towards remaining tasks

There are no remaining tasks.

6.3 Roadmap to exascale

Task	Estimated effort	Status
Alternative dynamical core option	10 PY	Ongoing
Extreme scaling with DAGs	12 PM (pilot)	In planning
DSLs for accelerator/host portability	2 PY (dynamics)	In planning
Scalable post processing and product generation	2 PY	In planning
Improved vectorization	1 PY	Ongoing
Scalable model startup	3 PM	Reported
Scalable grib_api initialization	2-3 PM	Reported
Improved memory scaling	3-6 PM	In planning
Further work on computation /communication overlap	2-3 PM	In planning

Table 6.3 Roadmap to exascale for IFS

6.3.1 Alternative dynamical core option

As discussed in Section 6.1, the scaling runs for the T_C3999 2.5 km model have shown that the communication cost for the spectral transform method in IFS is high. While overlapping computation and communication shows promising results, this appears to be less effective at scale as the computation scales better than the communication. This and the requirement to improve the performance of the IFS dynamics (improved conservation of mass) are driving the need to provide an alternative dynamical core option.

6.3.2 Extreme scaling with DAGs

Based on past experience, the current IFS model is expected to scale well up to a point where each thread has no less than 100 grid columns. The 2.5 km model has some 80 million grid columns, which implies an upper limit of scalability at around 800K threads, which is about 100 times less than the likely core count of an exascale system. Rather than speculate how an even higher resolution model would perform, we need to continue to explore ideas to improve the scalability of IFS at the petascale. Within CRESTA we have tested a small IFS kernel to understand how OmpSs tasking model can be used to overlap computation and communication. Such an approach is attractive for its simplicity, and moreover OmpSs can also be used to expose greater parallelism throughout IFS. An example could be to expose the independent matrix-matrix multiplies (DGEMMs) in each stage of the fast Legendre transform butterfly scheme. Another example could be to implement a radiation in parallel (or similar processes) scheme, and leave it to the DAG scheduler to execute 'radiation' threads in parallel with 'model' threads. In this approach we are alleviated from the complexity of managing separate MPI tasks for such processes and balancing their number with model MPI tasks [3]. To take this further we would propose a pilot port of IFS to use OmpSs, where the primary aim would be to significantly increase the number of threads of execution. Maximizing the number of cores (executing threads) per OmpSs process would have the benefit of reducing the effect of static load imbalance and also for an IFS model to be more resilient to the effects of system jitter. The use of a thread safe MPI or use of GASPI/GPI would allow more than one thread to be doing communication while other threads are performing computation tasks. In the longer term we would expect the OmpSs capabilities to be supported by a future OpenMP standard

6.3.3 DSLs for accelerator/host portability

The use of Domain Specific Languages (DSLs) will need to be explored to ease the porting effort to future accelerators (GPU/MIC) while continuing to maintain a single source version for IFS. Portability is an important requirement for IFS as it is jointly developed between ECMWF and Meteo France (where it is called ARPEGE) and derivatives of IFS being used in other weather centres. An example of a DSL used by MeteoSwiss in their local area COSMO model is described in [5].

6.3.4 Scalable post processing and product generation

Post processing is the process by which IFS model fields are gathered and written out in parallel to a file system, and subsequently archived to an archive store. Product generation takes these model fields and produces products (e.g. global fields or local area sections) that are disseminated to the ECMWF member states. At the exascale it will be prohibitively expensive to gather globally distributed fields on a regular basis (every model hour today), so alternative approaches must be found. A possible approach could be for a task to move its data to a shared memory location on its node, and leave it for some other non-time-critical processes to gather and post process on the fly. Storing raw data at the exascale may not be affordable, and a data reduction process may need to be used prior to archiving model fields.

6.3.5 Improved vectorization

Getting the best performance on the latest processor cores requires loops that vectorize. While this is mainly the responsibility of a compiler, sometimes directives are required to inform the compiler that particular loops are safe to vectorize. In some cases routines may require some refactoring to get the best vector performance.

6.3.6 Scalable model startup

In the CRESTA project we observed that model startup was scaling poorly, and using Vampir it was easy to see the main reason for this. For the 2.5 km model case startup took up to 12 minutes at about 200K cores on TITAN. The reason is simply that IFS uses a single task for reading the initial data (160GB @ 235 MB/s), and although the reader task sends this data (a multi megabyte logical record at a time using MPI non-blocking sends) to other tasks for decoding in parallel, the bottleneck remains the single reader. The proposed solution is to use a number of tasks to read the initial data in parallel which requires the originator of this data to do corresponding writes in parallel to separate files. The subsequent scatter operations of fields to the required task distribution are insignificant in cost, and in particular as this is done only once per file type (spectral, surface, upper air) at initialization today.

6.3.7 Scalable grib_api initialization

GRIB is the World Meteorological Organization standard for encoding meteorological data, and as part of the initialization phase of an IFS model we call a setup routine on all tasks that expect to use the grib_api interface to encode or decode data, which by default is all tasks. The problem with this setup routine is that it read tables (files) that are spread over 39 files which in total are less than 1 MByte. Reading such data from a single directory from 28K tasks on TITAN took about 10 minutes. A workaround was found to reduce the number of tasks that need to perform this initialization to about 1K, and further to hide this in the dead time that tasks are waiting for the initial data reads to complete. Of course, the correct solution should be for a single task to read the data and broadcast it to all tasks, taking under a second. This should all be hidden within the grib_api interface setup.

6.3.8 Improved memory scaling

As we double the number of tasks, we would expect our memory use to halve, which is not the case for an IFS model. This needs to be addressed, and to be revisited every time a new code cycle of IFS is produced. It would be interesting to see if Allinea DDT/MAP can shed some light on this memory scaling issue, but we already have some suspects. One such code area is the semi-Lagrangian scheme where a non-scaling data structure is the wide halo area surrounding the grid points that a task owns. The issue with this data structure it is that it is the leading dimension (with fields as the second dimension). Transposing these dimensions will not reduce the virtual space, but should reduce the number of real pages used particularly when only a small part of the wide halo is actually used (as is the case with the CRESTA SL optimization). The downside of the transpose is that all the interpolation routines would need to be substantially modified and not just swapping the dimensions of some array declarations.

6.3.9 Further work on computation/communication overlap

The current coarray support in Fortran2008 lacks the capability to allocate and deallocate coarrays for a subset of images. The next Fortran standard is expected to rectify this deficiency by providing support for a coarray team. With this capability, a coarray team can be created and a coarray can be allocated within this team, allowing greater flexibility, and avoiding the overhead and necessary synchronisation of global coarray operations that exist today. The same functionality already exists with GASPI/GPI, which can be used to provide a more portable language independent approach for computation/communication overlap.

6.4 Application performance and scalability

For an IFS model, it is critical that in operations it can run a 10 day forecast in under one hour, which equates to 240 forecast days per day. During the CRESTA project ECMWF have focused on the performance of IFS model resolutions that it would expect to be running based on past experience, a halving of model resolution every 8 years. A 10 km global model is currently planned to enter operations in 2015 on the newly installed Cray computers at ECMWF, two XC-30 systems each with 85,000 Intel Ivybridge cores. This would suggest the next IFS model resolution upgrade would be to a 5 km model in 2023-24 and thereafter to a 2.5 km model in the early 2030's. It is possible that by the early 2030's ECMWF would have an Exascale system if the projected growth of "Top500" systems continues on its near Moore's law trajectory (this is a big assumption). Having access to HECToR in 2011/12 and TITAN in 2013/14 has allowed ECMWF to run the 10 km, 5 km and 2.5 km IFS models at scale for short 3 or 6 forecast hour periods, and measure performance as shown below.

6.4.1 IFS 10 km global model

Figure 6.3 shows how scaling has improved over the CRESTA project for the 10 km IFS global model. The final OCT-14 scaling improvement was obtained by simply reducing the frequency of producing global grid point and spectral norms. These norms have no effect on model results and are only useful to ECMWF scientists during model development. While these norms are coded using an efficient 2D parallelization scheme, these runs are a clear indicator that their use has a negative effect at scale. Figure 6.4 shows the performance gains that are due to the coarray optimizations. As the operational requirement is 240 forecast days per day, it can be seen that there is no scaling issue at the core count (8K cores) to achieve this performance, and the improvement due to the coarray optimizations is about 8 percent (on TITAN).

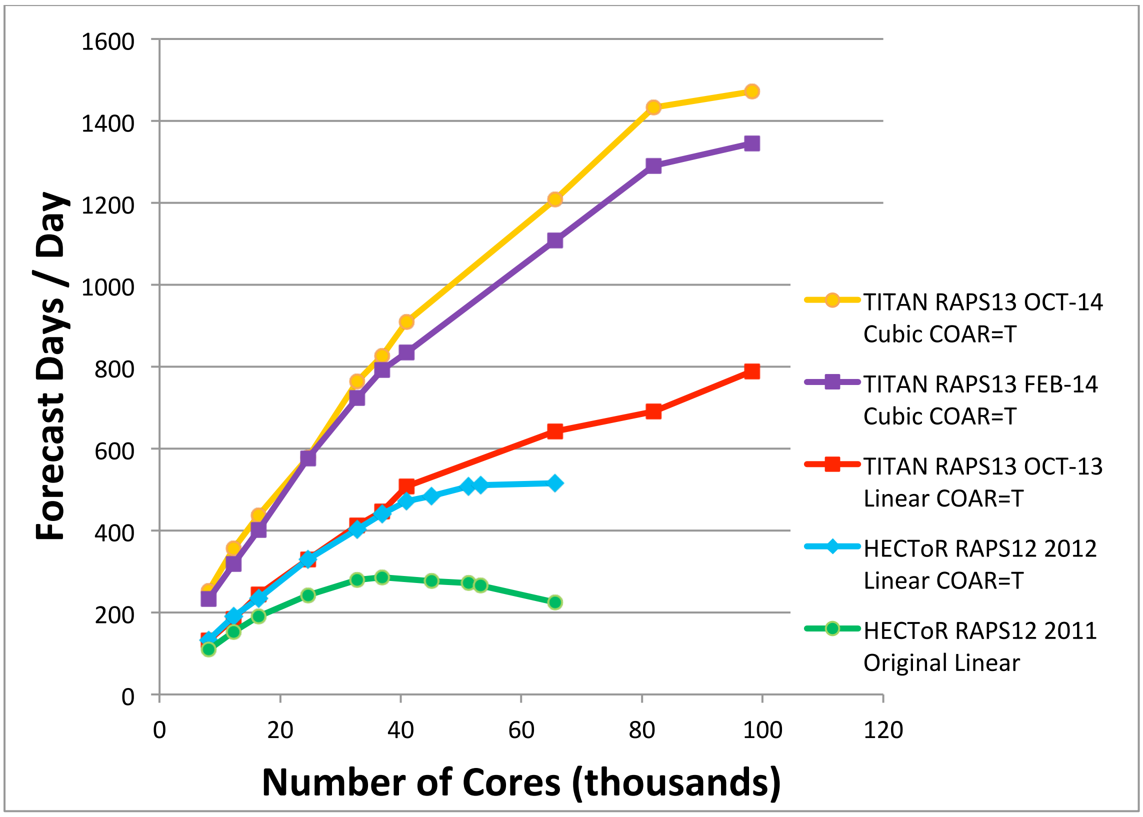


Figure 6.3 10 km / L137 global IFS forecast model performance, RAPS12 (CY37R3, on HECToR), RAPS13 (CY38R2, on TITAN)

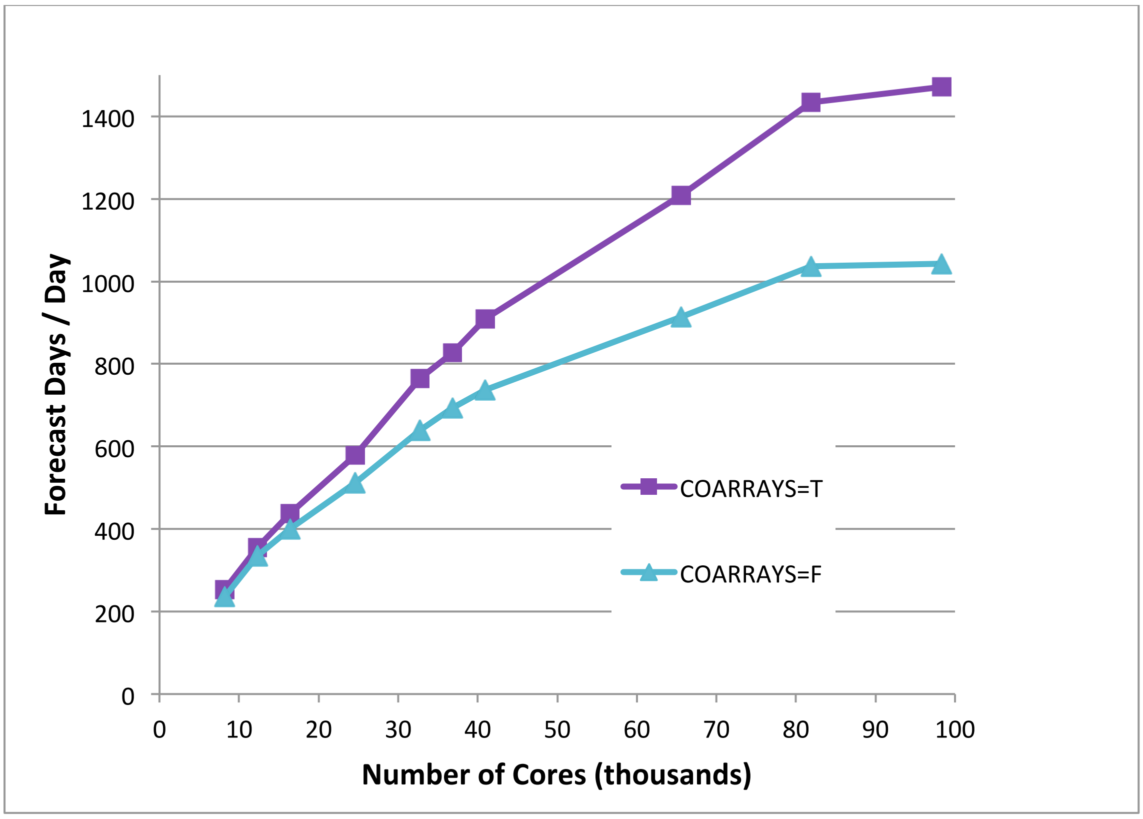


Figure 6.4 10 km / L137 global IFS forecast model performance, showing performance improvement due to coarray optimizations

6.4.2 IFS 5 km global model

Figure 6.5 shows the performance of a 5 km model running on TITAN and ECMWF's Cray XC-30. It is clear from the detailed timers in IFS that the XC-30 is both faster per core (Intel Ivybridge versus AMD Opteron) and has improved communications performance over TITAN (Cray Aries versus Cray Gemini interconnect). Yes, the XC-30 would meet the 240 forecast days per day operational requirement, but the 25K cores needed would be too high a fraction of the total 85K cores (on one cluster) to run the rest of the operational suite (ensemble prediction system, ensemble data assimilation) and other workloads.

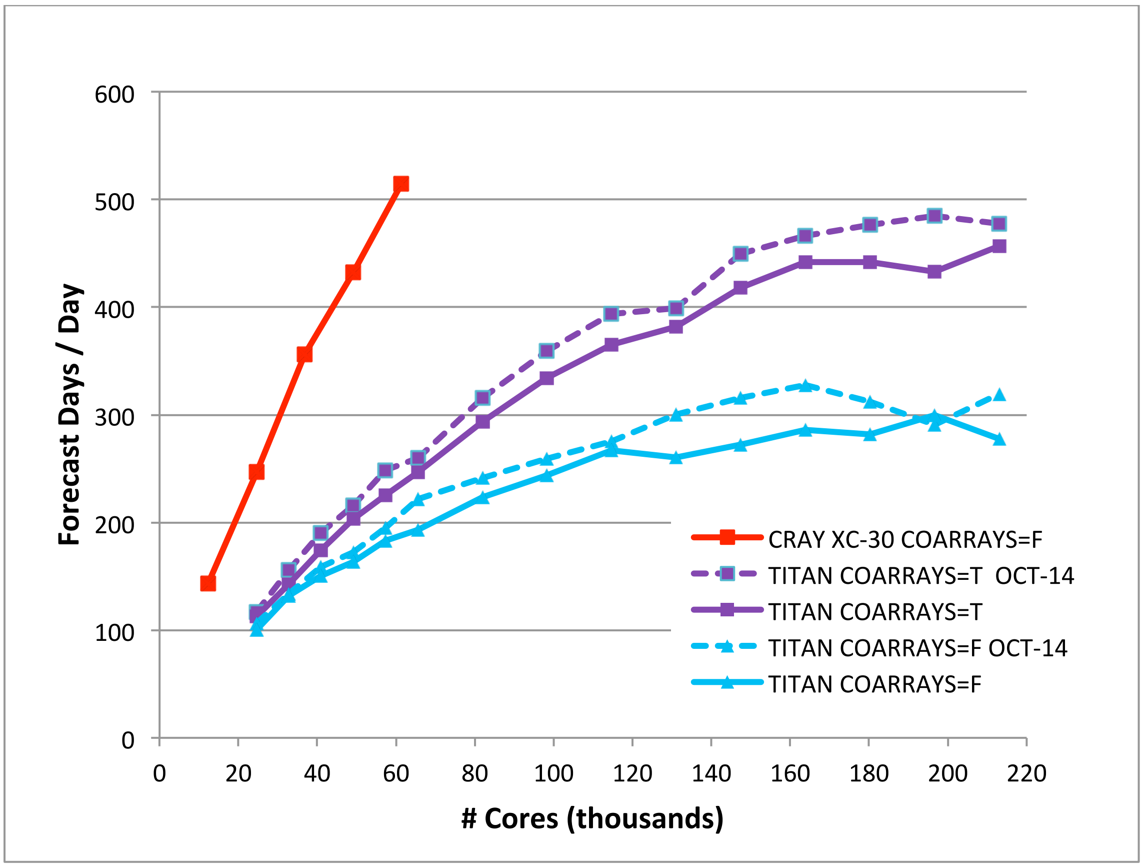


Figure 6.5 5 km / L137 global IFS forecast model performance

The figure also shows the improvement in performance on TITAN at scale by reducing the frequency of spectral and global norms (the dashed lines - labelled OCT-14).

So which of the coarray optimizations had the largest effect on performance?

Figure 6.6 shows how the performance attributed to the coarray optimizations are distributed as a percentage of their total. By far, the most improvement came from the semi-Lagrangian optimization at about 50%, then the Legendre optimizations, and finally the Fourier optimizations.

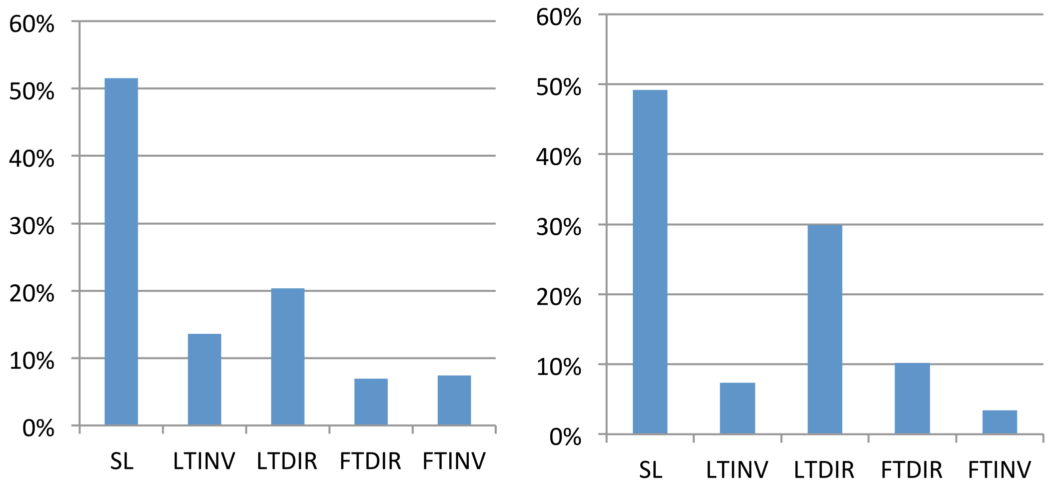


Figure 6.6 Coarray optimization distribution for Tc1999 on Titan. Left picture: using 49,152 cores, Right picture: using 163,840 cores

6.4.3 IFS 2.5 km global model

Figure 6.7 shows the performance of a 2.5 km model running on TITAN and ECMWF's Cray XC-30.

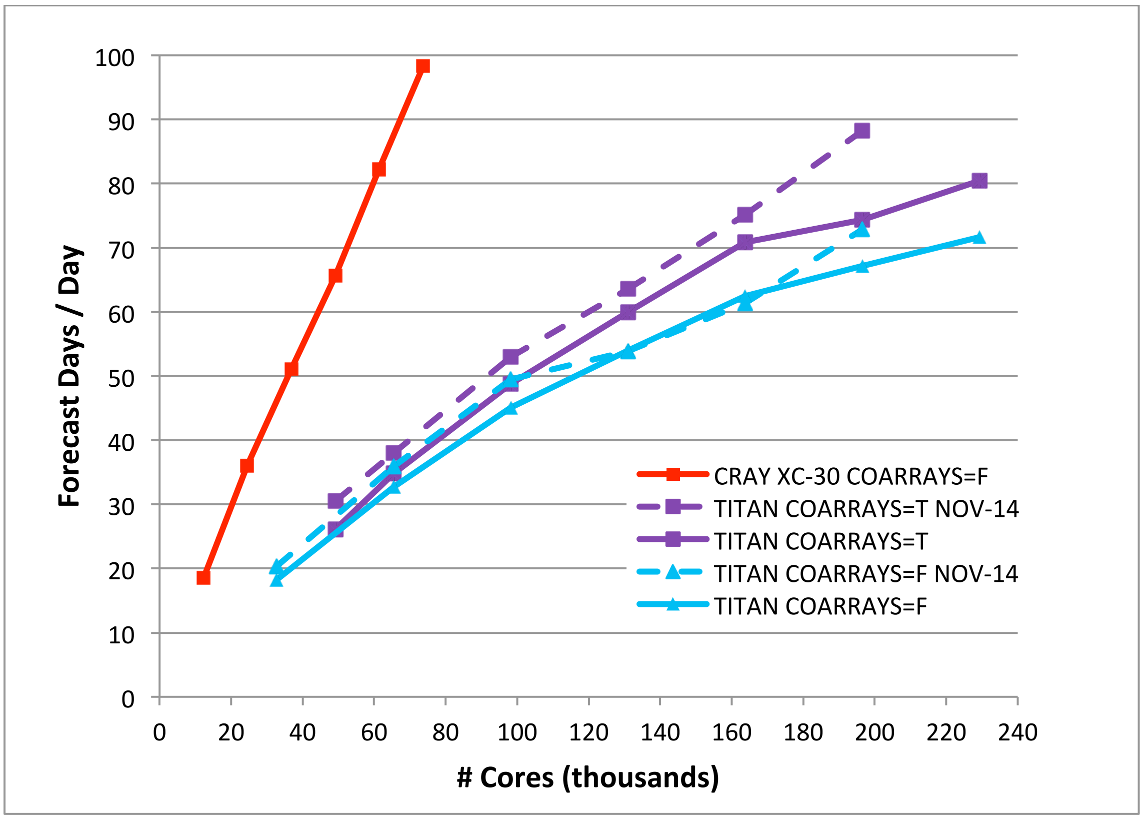


Figure 6.7 2.5 km / L137 global IFS forecast model performance

The latest runs (labeled NOV-14) show a marked improvement in performance at scale by removing spectral norm and grid-point norm computations and communications, except for the final time-step needed for the correctness check. These norms are purely diagnostic and have no effect on model results. While these norms are useful during the development process, they serve no purpose for operational runs. Unfortunately, the NOV-14 model runs at 229K cores both encountered a huge-page problem on TITAN, which has been reported to support staff at ORNL. It is suspected that this related to memory fragmentation and reduced huge-pages on some nodes. Note that none of these runs are getting close to the desired 240 forecast days per day.

6.4.4 Efficient weak scalability, IFS 10 km global model

Figure 6.8 shows the weak scaling of an IFS model on a Cray XC-30 system. The data point labeled 240 is for a 10 km model (144 nodes), 218 a 5 km model (821 nodes) and 177 a 2.5 km model (7122 nodes). As the ECMWF Cray XC-30 system has a maximum of 3500 nodes, the 2.5 km performance at 7122 nodes was extrapolated. The number of nodes used for the 3 model cases was simply scaled by the number of model grid-points and the time-step used, and does not include any non-linear factors in the spectral transform method or the radiation grid used.

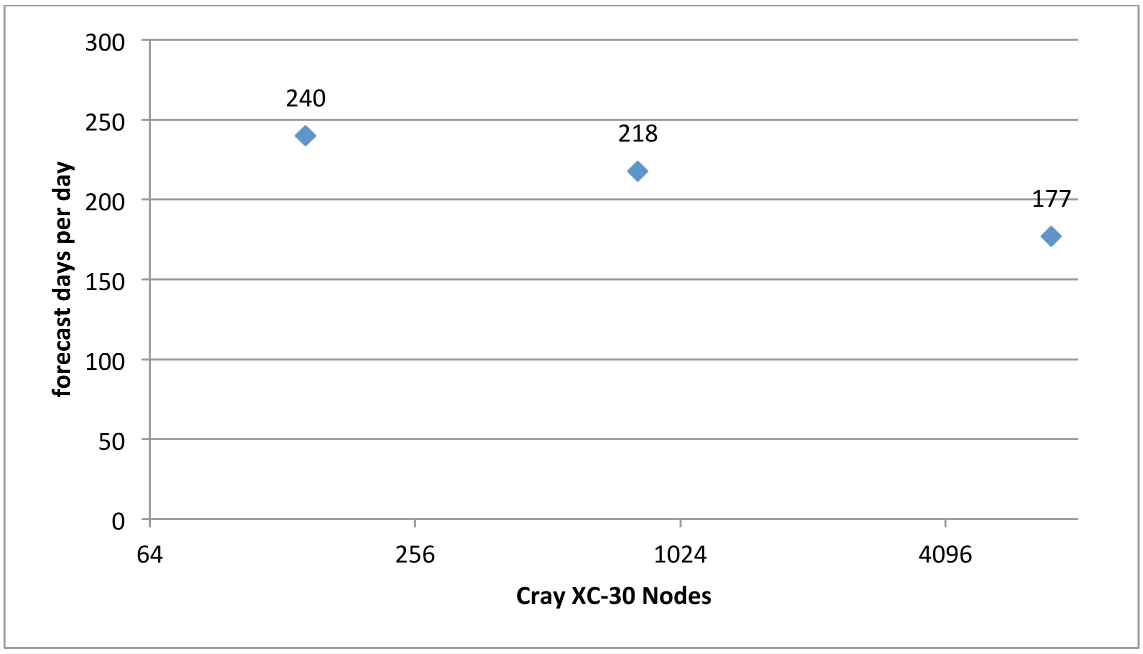


Figure 6.8 Weak scaling of IFS model on a Cray XC-30

It is clear from the weak scaling that an IFS model requires continued scalability improvements to get closer to the ideal weak scaling of a horizontal line.

6.4.5 Summary of the performance improvements achieved

Table 6.4 shows the performance improvements realized in the CRESTA project for a 10 km global 137 level hydrostatic IFS model case using 45,056 cores on HECToR and TITAN (both use AMD Interlagos cores). The performance measure is Forecast Days per Day (FD/D), where the operational requirement for a 10 day deterministic forecast is one hour or 240 FD/D. This model case is expected to enter operations at ECMWF in 3Q2015.

Code version	Compiler Release/System	10 km model FD/D	Relative Performance
RAPS12 (CY37R3) base, linear grid, TSTEP=450s	8.0.3 HECToR	277	1.00
MPI optimizations to wave model	8.0.3 HECToR	356	1.29
new compiler release, improved compiler opts	8.0.6 HECToR	419	1.51
All coarray optimizations (LT, FT, SL)	8.0.6 HECToR	485	1.75

RAPS13 (CY38R2) base	8.1.5 TITAN	500 est.	1.80 est.
Using cubic grid (still 10km global grid), TSTEP=600s	8.2.2 TITAN	880 est.	3.17 est.
Final runs OCT-14 with reduced norms	8.3.0 TITAN	925	3.34

Table 6.4 Evolution of IFS 10 km L137 model performance using 45,056 cores on HECToR and TITAN

It is clear that IFS model scalability has improved substantially during the CRESTA project, and has shown encouraging scalability at the petascale. However, getting an IFS model to perform well at the exascale will require many further developments as outlined in the roadmap in section 6.3. What is clearly also required is ongoing hardware developments to keep power costs to acceptable levels and a software stack that provides for performance, portability and maintainability of large applications such as IFS.

6.5 References

[1] Smolarkiewicz, Piotr K.; Kühnlein, Christian; Wedi, Nils P. "A consistent framework for discrete integrations of soundproof and compressible PDEs of atmospheric dynamics." J. Comput. PhysicsJ. Comput. Physics 263, 185-205(2014) doi: /10.1016/j.jcp.2014.01.031

[2] Marcin J. Kurowski, Wojciech W. Grabowski, and Piotr K. Smolarkiewicz, "Anelastic and Compressible Simulation of Moist Deep Convection." J. Atmos. Sci., 71, 3767–3787. 2014, doi: /10.1175/JAS-D-14-0017.1

[3] Mozdzynski, G., and J.-J. Morcrette, Reorganization of the radiation transfer calculations in the ECMWF IFS. ECMWF Technical Memorandum No.721, April 2014.http://old.ecmwf.int/publications/library/ecpublications/_pdf/tm/701-800/tm721.pdf

[4] Christiane Jablonowski , "Idealized Dynamical Core Test Cases for Weather and Climate Models", http://www-personal.umich.edu/~cjablono/dycore_test_suite.html

[5] Gysi, Tobias; Fuhrer, Oliver; Osuna, Carlos; Cumming, Benjamin; Schulthess, Thomas, "STELLA: A domain-specific embedded language for stencil codes on structured grids", EGU General Assembly 2014, 2014EGUGA..16.8464G.

[6] OmpSs home page, https://pm.bsc.es/ompss.

7 Nek5000

Nek5000 [7] is an open-source code for the simulation of incompressible flow in complex geometries. The discretization is based on the spectral-element method (SEM) which combines the higher-order accuracy from spectral methods with the geometric flexibility of finite element methods.

Nek5000 is written in mixed Fortran77/C and designed to employ fully large-scale parallelism. The code has a long history of HPC development. Recently the large-scale simulations were successful performed on the Cray XE6 system at PDC, KTH with 32,768 cores [8] and on the IBM BG/P Eugene with 262144 cores [9]. An overview of the capabilities and recent developments within the Nek5000 community is given in [10].

7.1 Summary of the previous roadmaps

Task	Scheduled date	Status
Investigate existing code architecture	M18	Completed
Implement error estimator and initial refinement code	M24	Completed
Adaptive refinement development	M18	Ongoing
Implement load balancing using existing Nek5000 tool suite	M30	Ongoing
Undertake test and development on large scale applications	M30	Completed
OpenACC acceleration of Nek5000	M27	Completed

Table 7.1 Summary of the previous roadmaps for Nek5000

Investigate existing code architecture

The aim of this task is to gain a fundamental understanding of most aspects of the implementation of NEK5000 with a special attention to the large-scale simulation of incompressible flow.

Implement error estimator and initial refinement code

Adaptive Mesh Refinement (AMR) requires identification of the regions in the flow with significant error. Error estimators based on the expansion of the solution in the basis of Legendre functions were successfully implemented in NEK5000.

Adaptive refinement development

AMR gives possibility to increase the accuracy of numerical simulations with minimal computational cost. There are two ways of introducing AMR: adaptive p-refinement, i.e. increasing polynomial order in individual elements, and adaptive h-refinement, i.e. splitting the element into smaller one. We have discarded p-refinement in favor of h-refinement, due to its flexibility. To manage variable grid structure we used p4est [1] library. We have integrated NEK5000 with p4est library adding all the tools necessary for mesh regeneration and redistribution.

Implement load balancing using existing Nek5000 tool suite

NEK5000 obtains full scaling using static load balancing based on initial element distribution. After introducing AMR the load balancing become an important issue, as the grid structure changes during the simulation. We implemented dynamical grid partitioning using the ParMetis [2] library.

Undertake test and development on large scale applications

By using the developed software environments we conducted large-scale model simulations of the heat transfer problem.

OpenACC acceleration of Nek5000

The objective of this task was to enable the use of Nek5000 on a massively parallel hybrid GPU/CPU system. To this end, we first implemented an OpenACC version NekBone benchmark, which is a simplified version of Nek5000. The knowledge gained from the Nekbone implementation was used to perform the OpenACC acceleration of the full Nek5000 code. In addition, the task was used to assess the viability of hybrid exascale simulations and the status and the performance of the current OpenACC compilers.

7.2 Achievements towards remaining tasks

Task	Achievement
Adaptive refinement development	Adaptation of the pressure preconditioners for AMR

Table 7.2 Achievements towards remaining tasks from the previous roadmaps for Nek5000

Adaptive refinement development

We have implemented h-type refinement into NEK5000 using p4est library as grid manager. It allows us to solve diffusion equation with time independent velocity field on the dynamically changing grid using adopted for AMR conjugated gradient method. However, integration of full incompressible Navier-Stokes solver in Nek5000 with AMR requires significant modification of the pressure solver which is the most communication extensive part of the code. To assure fluid incompressibility standard methods like conjugated gradient method would require number of iteration proportional to the grid point number to converge. This limits their usage to the relatively small problems only. To reduce the number of iterations required, specialized preconditioners are needed. However, their adaptation to AMR requires additional algorithm development, which we will not be able to finish within CRESTA.

7.3 Roadmap to exascale

Task	Estimated effort	Status
Correcting load balancing using existing Nek5000 tool suite	3 PM	Ongoing
Improving multi-GPU communication with OpenACC	6 PM	In planning

Table 7.3 Roadmap to exascale for Nek5000

7.3.1 Correcting load balancing using existing Nek5000 tool suite.

NEK5000 obtains full scaling using static load balancing based on initial element distribution. After introducing AMR the dynamic load balancing become an important issue, as the grid structure changes during the simulation. In cooperation with WP5 we implemented dynamic grid partitioner using standard libraries for graph bisection adopting partitioning from scratch strategy. This strategy allows for highest possible quality of the mesh distribution, but does not take into account partitioning and communication costs. Our tests performed with ParMetis show the partitioning time to grow quickly with number of processors and to be dominant in the runs with less than 10 elements per core. That is why we investigate possible improvements testing different repartitioning strategies. This work is performed in cooperation with WP5.

7.3.2 Improving multi-GPU communication with OpenACC

With a multi-GPU setup, the gather-scatter operator and the associated MPI communication can be improved. In the CRESTA project, the original gather-scatter operator was split into two parts. First a local gather on the GPU is performed, followed by the transfer of the boundary values at the interfaces of the domain. Then the boundary values need to be copied to a local CPU memory, communicated via network to the memory of another CPU, and then transferred back a memory of a remote GPU to finally carry out a local scatter on the GPU. This approach allows a considerable reduction in the amount of data to be moved from the GPU and CPU memory and vice versa.

In spite of the reduction in the amount of data transferred, the additional transfers between the host and accelerator have an effect on the achievable performance. We estimate that techniques such as overlapping of GPU kernels with host/accelerator memory transfers or using direct communication between accelerators via RDMA would increase the performance of the OpenACC version of Nek5000. This is will be part of future research.

7.4 Application performance and scalability

The original (conformal meshes; no accelerator support) version of Nek5000 scales up to 10^6 processes with parallel efficiency 0.6 on ALCF BG/Q Mira (see Figure 7.1). This good scaling is achieved by proper construction of the global communication and code simplicity.

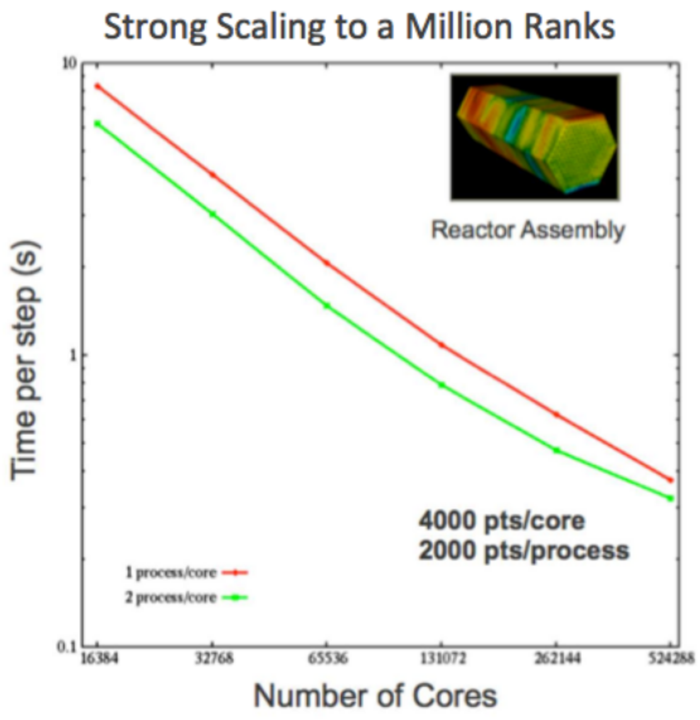


Figure 7.1 Strong scaling of original Nek5000

There are two main tasks related to Nek5000 development in CRESTA: code adaptation for GPGPU accelerators with OpenACC and the implementation of h-type Adaptive Mesh Refinement (AMR). Both of these tasks require significant modification of the global communication pattern that may have a negative impact on the strong scaling of the code.

To check the effect of the modifications, we test the modified code on different model problems and run exemplar scientific simulations for **GPGPU** and **AMR** scalability tests.

7.4.1 GPGPU: evaluation of application performance and scalability

The Titan supercomputer has been used to study the performance of OpenACC accelerated Nek5000 on GPU systems. Titan is a Cray XK7 supercomputer, containing 18,688 AMD Opteron 6274 16-core CPUs and 18,688 Nvidia Tesla K20X GPUs.

GPGPU: Model problem

NekBone is configured with the basic structure and user interface of the extensive Nek5000 software. NekBone exposes the principal computational kernel to reveal the essential elements of the algorithmic-architectural coupling that is pertinent to Nek5000, (More information about NekBone can be found in [11].

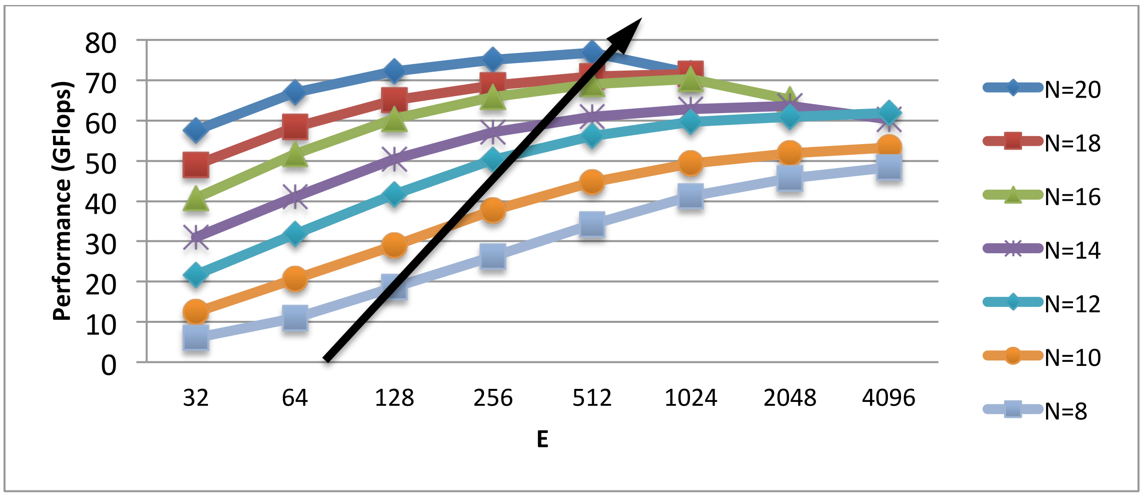


Figure 7.2 The performance of NekBone on a single GPU varying the number of elements (E) and the order of the polynomial (N). By varying these two parameters the changed computational workload on the GPU is changed with which affects the computational performance.

Consequently, the results from investigating the performance and profiling for NekBone can be directly applied to Nek5000. NekBone solves a standard Poisson equation using the spectral element method with an iterative conjugate gradient solver. The computational performance critically depends on the computation workload on the GPU. The larger the number calculations that are completed on the GPU, the higher the performance is. This is clear from Figure 7.2 which shows the performance of NekBone on a single GPU varying the number of elements (E) and the spectral order (N). By increasing these two values, we increase the trip-count of the four nested loops in the matrix-matrix multiplications. Better performance is achieved with a high number of elements and a high order polynomial.

Different compilers and versions also affect the performance of NekBone. Figure 7.3 shows the performance in Gflops on a single GPU with the Cray CCE OpenACC, PGI OpenACC and PGI CUDA Fortran compilers, respectively. The best performance (of around 75 Gflops) was obtained with the latest version of PGI V14.10.0. The number of elements used in these test was 1024.

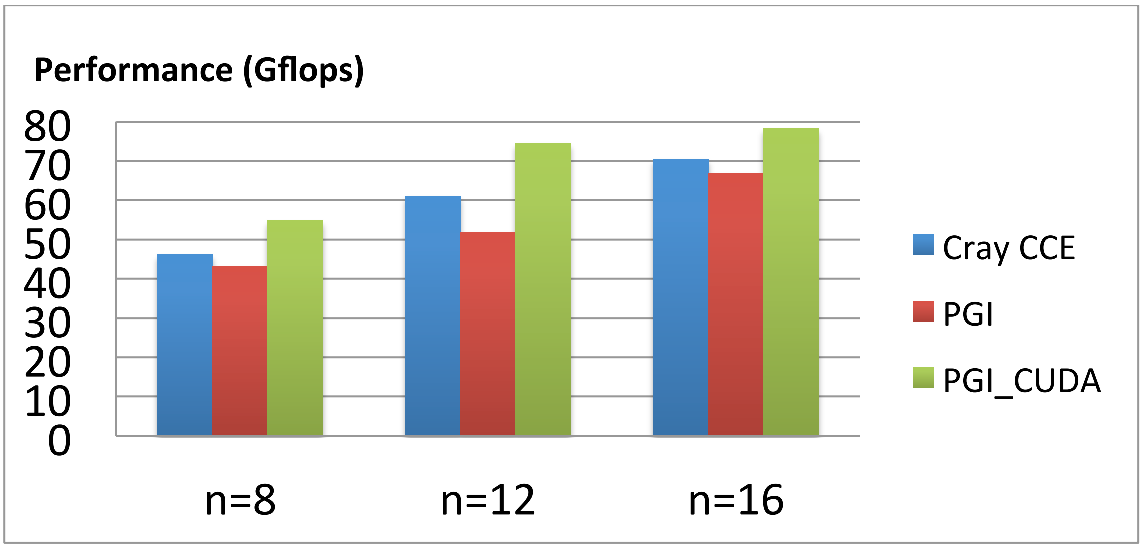


Figure 7.3 The performance of NekBone on a single GPU with different compilers. 1024 elements were used. Cray CCE: Cray CCE v8.4.0 compiler; PGI: PGI v14.9.0 compiler; PGI_CUDA: PGI v14.9.0 with CUDA Fortran.

GPGPU: Exemplar scientific simulation

The study of turbulent pipe flows is closely related to finding the relationship between the average flow velocities and the friction coefficient. The flow of fluid in pipes with circular cross-section is frequently encountered in a variety of environmental, technical and even biological applications. Typical examples of pipe flows can be found in urban drainage systems, the transport of natural gas or oil in the energy sector (i.e. pipelines) or the flow of blood in veins and arteries. Simulations can help us to find methods to solve problems related to pipe flow, such as drag reduction. Thus, the understanding of flow physics in pipes has a direct and significant substantial impact on everyday life [6].

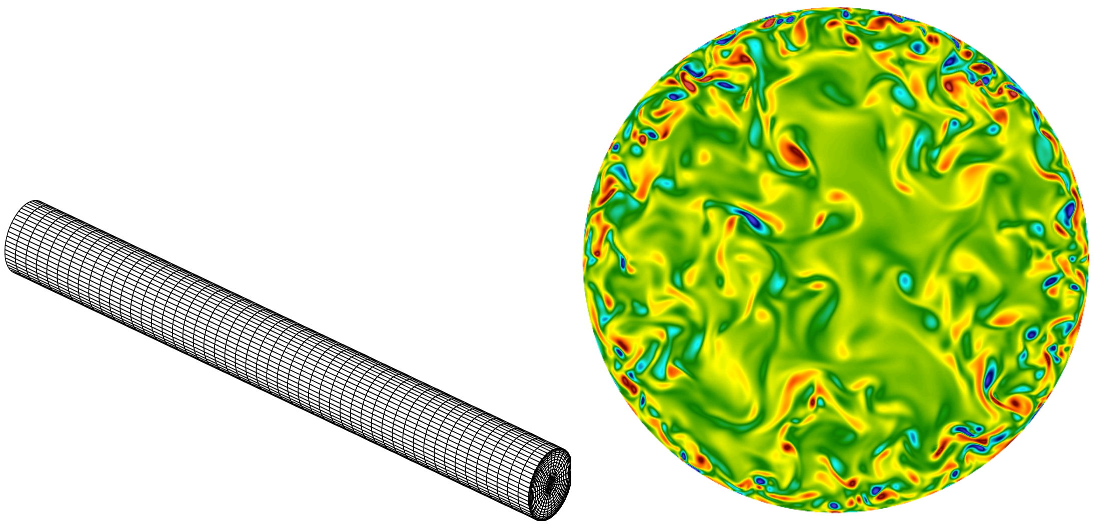


Figure 7.4 Mesh geometry (left) and the instantaneous axial vorticity, displayed for Re=190000 (right)

In Nek5000, Direct numerical simulation (DNS) is employed to numerically solve the Navier-Stokes equations. In order to capture all the features of an eddy in the flow field, the computational domain should be larger than the structures in the flow to resolve the smallest scale in the turbulent eddies. Consequently, the computational cost of performing a DNS including all scales grows by Re^3, where Re is the Reynolds number. The highest Reynolds numbers for a pipe flow can reach $5.3x10^5$. With such a high computational costs, only a substantial increase in the available computing power makes it possible to fully resolve numerical solutions of a truly turbulent flow.

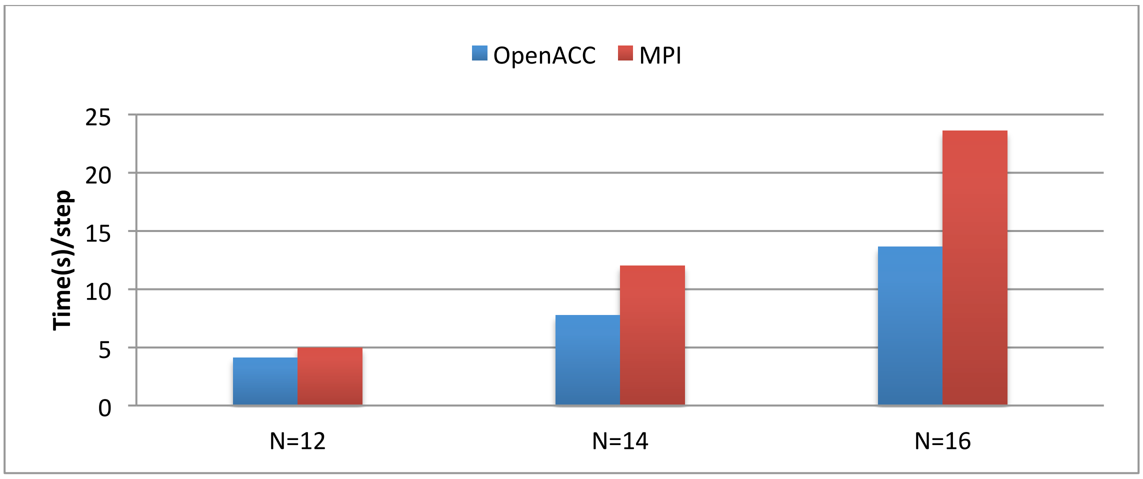


Figure 7.5 The execution time per iteration with different orders of polynomial using the CG linear solver and Schwarz preconditioner. 1024 elements were used on a single GPU. OpenACC: single GPU; MPI: single node with 16 cores.

A Nek5000 simulation of the flow in a straight pipe with 400 elements was used to test the performance of the Nek5000 code with OpenACC acceleration. The execution time per iteration with different orders of polynomial using the CG linear solver and Schwarz preconditioner are compared in Figure 7.5. The speed-up achieved using OpenACC directives is 1.74 with a16th order polynomial on a single GPU compared to single node with 16 CPU cores.

7.4.2 AMR: evaluation of application performance and scalability

All test runs on AMR were performed on a Cray XE6 system Lindgren at KTH. Lindgren computer has a total of 1516 nodes with dual AMD Opteron 12-core 2.1 GHz "Magny-Cours" processors (36384 cores in total).

AMR: Model problem

Our model problem is based on the convected-cone example introduced by Gottlieb and Orszag [3], which is the passive scalar transport problem. In the original problem a unit-height cone with a base-radius of 0.1 centered at (x,y)=(0, 0.25) is subjected to plane rotation according to time independent velocity field **v**=(y-0.5,0.5-x). We adopted this example to 3-dimensional simulations evolving a sphere-shape (strong scaling) or cylinder-shape (weak scaling) cone according to energy equation in Nek5000. Spectral error estimator identifies discontinuities in the initial condition increasing grid resolution at the edge and the center of the cone (see Figure 7.6).

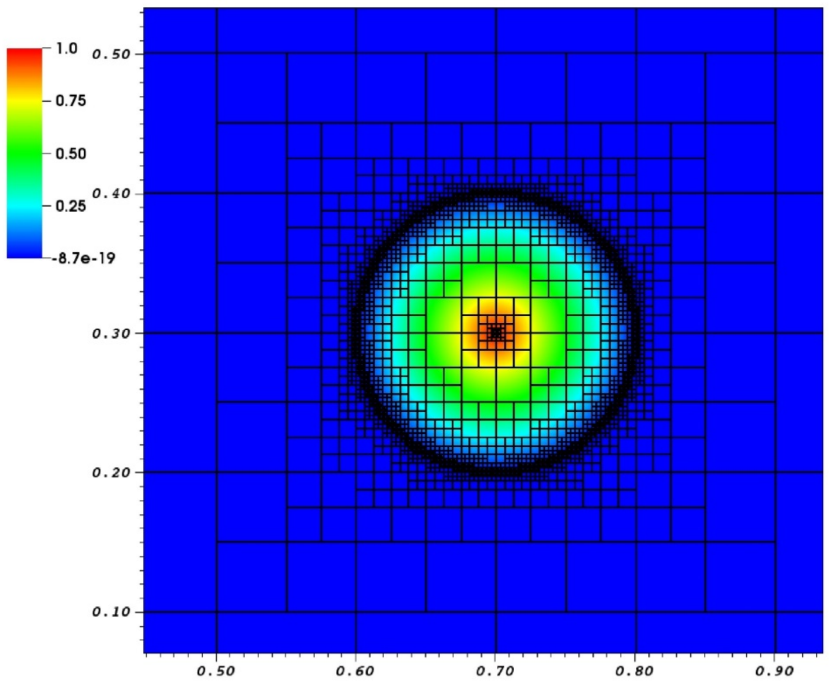


Figure 7.6 Two–dimensional cut through the domain of the convected-cone problem showing the grid structure (black squares) and the passive scalar profile (color scale). Each element (3D cube depicted by a square) corresponds to the mesh of 12X12X12 grid points.

In our runs we used 6 and 5 refinement levels for strong (sphere) and weak (cylinder) scaling tests, respectively. It corresponds to 33864 elements for strong scaling tests and 117192 elements for weak scaling test at 2048 cores. Scaling tests were performed for number of cores being power of 2 ranging from 2048 up to 32768. The global element number in the weak scaling tests ranges from 117192 up to 1875072. In all the runs the polynomial order was set to 11 and the graph bisection was performed by ParMetis. Note that the global number of elements in some of the strong scaling tests is not constant for different processor numbers due to the fact, that p4est performs de-refinement of the 8 children elements into the single parent element only if all the children elements reside on a single process. That is why the global number of elements slowly grows with the number of processors and the run performed on 32768 cores has in average about 6% elements more than the run performed on 2048 cores.

In all the tests performed we follow the advected features in the flow (the cone). This requires continuous adjustment of the mesh and does not converge to any time independent grid structure. Such a strategy is not applicable to stability calculations, where instead of individual flow structures the sensitive regions in the flow have to be identified and resolved. On the other hand, following the advected features of the flow allows us to increase the frequency of grid modification and to study possible limitations of the method. In presented runs with grid adaptation turned on the mesh was regenerated every 50 Nek5000 steps.

In all tests we study performance of different tools used in simulations focusing on two major stages: non-conformal Nek5000 calculation (**SOLVER**) and mesh adaptation phase (**AMR**). AMR phase consists of number of operations like identification of regions for refinement/de-refinement (**ESTIMATOR**), generation of the new grid performed by p4est (**P4EST**), mesh partitioning performed by ParMetis (**PARTITION**), sorting and transferring elements between processors (**TRANSFER**) and resetting Nek5000 solver on the new mesh (**RESTART**). The last operation consists of e.g. generation of new global communicators for direct stiffness summations, which require global ordering of the grid points done by p4est. Many of these operations are communication extensive and do not perform enough calculations to overlap communication. We will discuss performance all those operations in following sections.

AMR: Exemplar scientific simulation

We do not present here any of the scientific simulations due to the lack of proper, adopted for AMR pressure preconditioner, which limits the size of studied problem and does not allow to perform scaling tests. It is related to the fact that integration of full incompressible Navier-Stokes solver in Nek5000 with AMR requires significant modification of the pressure solver, which is the most communication extensive part of the code. To assure fluid incompressibility, standard methods such as conjugate gradient method would require the number of iterations proportional to the grid point number in order to converge. This limits their usage to relatively small problems only. To reduce CG iterations, specialized preconditioners are being used. However, their adaptation to the AMR framework requires additional algorithm development, which we will are unable to finish within CRESTA.

7.4.3 Efficient strong scalability, model problem

GPGPU: Model problem

For the test on Titan, 16th-order polynomials were used for a total of $8.59 \cdot 10^9$ points.

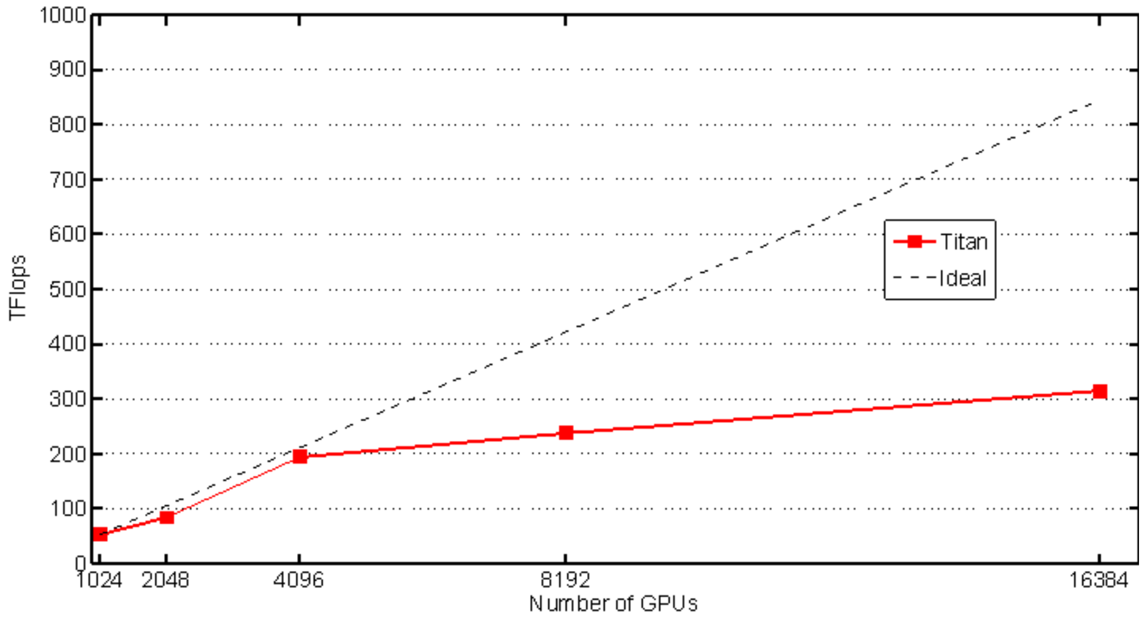


Figure 7.7 The strong scalability of NekBone. Total 2,097,152 elements and 16th order of polynomial are used.

Figure 7.7 shows the NekBone strong scaling performance, measured in Tflops, with up to 16,384 GPUs as a solid red color, while the black dashed line represents the ideal strong scaling. The parallel efficiency with 4096 GPUs was 91.9% compared when using 1024 GPUs. However the efficiency reduced significantly when going from using 4096 to 16,384 GPUs. In order to get better performance we should use as many elements per node as we can fit into GPU memory (6GB for the Kepler K20X card).

AMR: Model problem

The starting point of our discussion is performance test of the non-conformal version of Nek5000 (SOLVER) developed within CRESTA without taking into account mesh adaptivity. The major differences between non-conformal and conformal (original one) Nek5000 versions are the direct stiffness summation operator (related to global communication), which is more complicated for the non-conformal version, and the initial mesh partitioning, which in non-conformal case is performed by ParMetis. In the conformal version grid partitioning is performed by Nek5000 native software and the element imbalance cannot exceed 1. On the other hand element imbalance in ParMetis is controlled by the imbalance tolerance parameter ε, which value closer to 1.0 should give better balancing of the partitions.

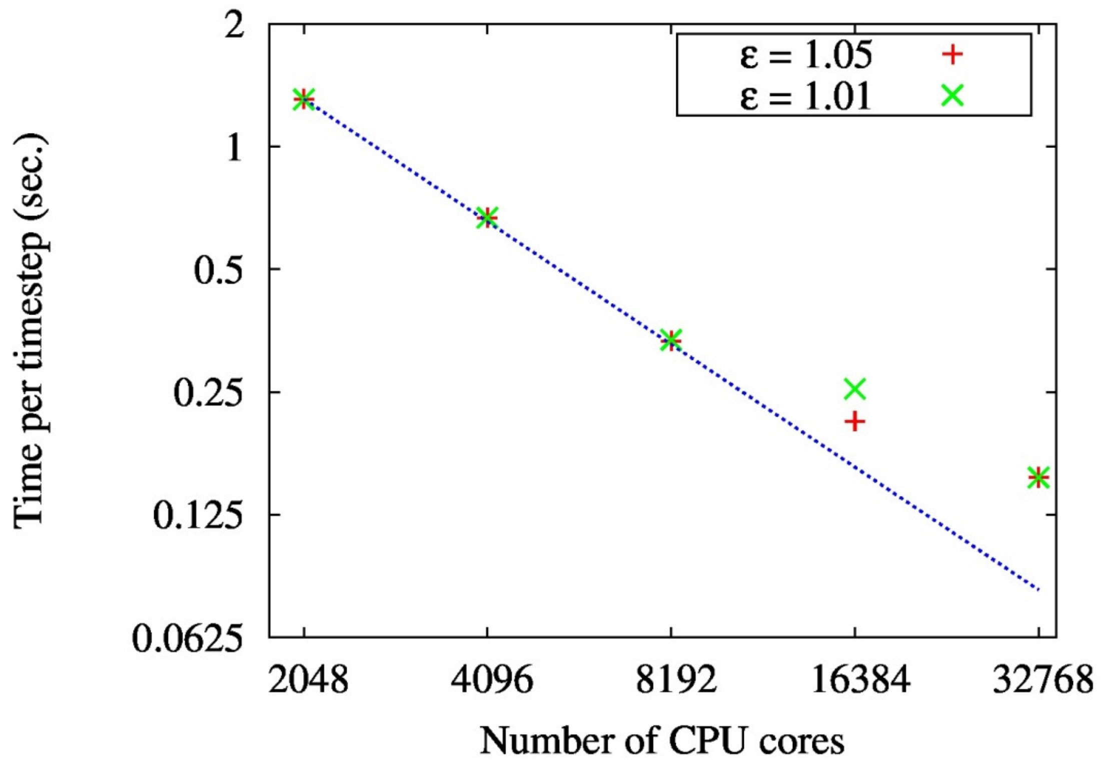


Figure 7.8 Strong scaling of non-conformal Nek5000 without mesh adaptivity for the imbalance tolerance ε equal 1.05 and 1.01. Blue line gives ideal scaling.

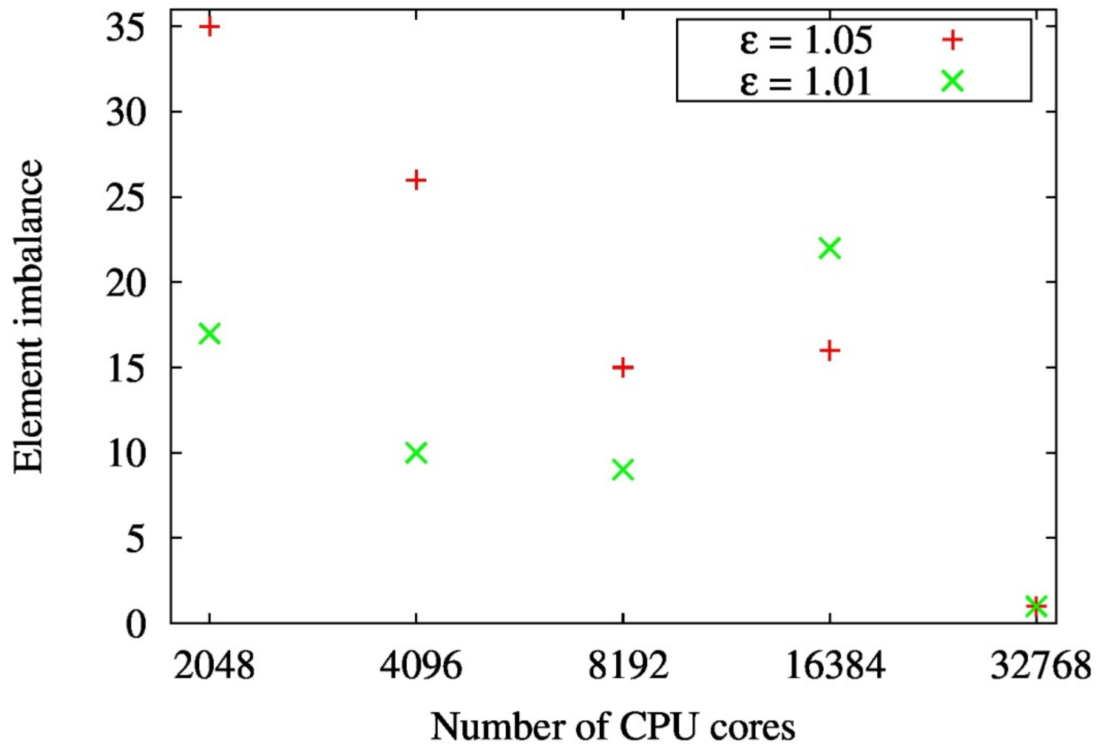


Figure 7.9 Elements imbalance as a function of processor number for the imbalance tolerance ε equal 1.05 and 1.01.

To check performance of the non-conformal Nek5000 version we executed number of simulations with constant, non-conformal grid structure (grid adaptivity turned off) and varying ε. In this case the mesh is generated and redistributed only once during initialization phase. The recommended value of ε is 1.05 and we did number of runs with ε equal 1.05, 1.01 and 1.001 founding no difference between results of ε=1.01 and ε=1.001 simulations.

The average time per time step and the element imbalance in the simulations is presented in Figure 7.8 and Figure 7.9. Unlike the native Nek5000 grid partitioner, ParMetis can give significant element imbalance, which for some cases is comparable with or even bigger than the minimum number of elements per core. For example in the case of the number of processors NP=16384 and ε=1.01 the element imbalance is equal 22 and number of elements per core ranges from 12 up to 34. This has strong impact on the code scalability reducing parallel efficiency as can be seen in Figure 7.8. Note that the lower value of ε does not always guarantee more balanced partitions. The lowest element imbalance was achieved for NP=32768 independently on value of ε giving parallel efficiency equal 0.55. This parallel efficiency was reached for about 15000 grid points per processor (corresponding to about 10 elements) setting minimal amount of work necessary for reasonable scaling.

Next investigated quantity is the time necessary to perform grid partitioning presented in Figure 7.11. Most of this time is spent in ParMetis and is dependent on the strong scalability of this library. Figure 7.10 shows slow increase of the partitioning time with the number of processors NP for NP≤16384 and next rapid jump (about 40 times) for NP=32768. As the number of partitions grows with the number of processors, the slow increase of the partitioning time is expected, however we are unable to explain the size of the jump for biggest NP. It is certainly related to decreasing number of graph nodes per core with growing NP and sets minimal amount of elements per core for ParMetis to achieve reasonable scaling. In our case this limit is about 20 elements per core, which is higher than the limit for the solver itself and shows Nek5000 solver to scale better than ParMetis library for studied problems.

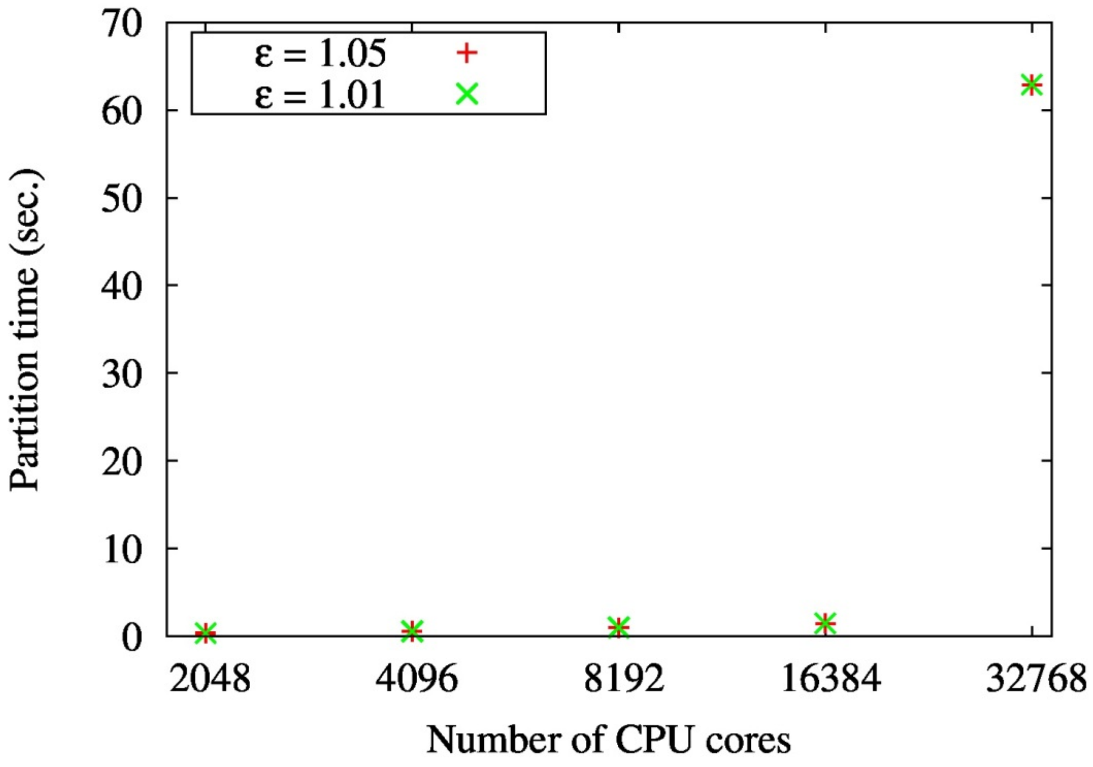


Figure 7.10 Partitioning time as a function of processors number.

Limited scalability of ParMetis is even more prominent in the tests with mesh adaptivity turned on, as the mesh partitioning is performed frequently. It is clearly visible in Figure 7.11 presenting the percentage of time spent in Nek5000 solver (SOLVER) and grid adaptation stage (AMR), and in Figure 7.12 showing average time per timestep including both SOLVER and AMR stages. The percentage of time spent in AMR stage is constantly growing with growing CPU number and for NP=32768 reaches about 90% causing significant increase of the average time per timestep.

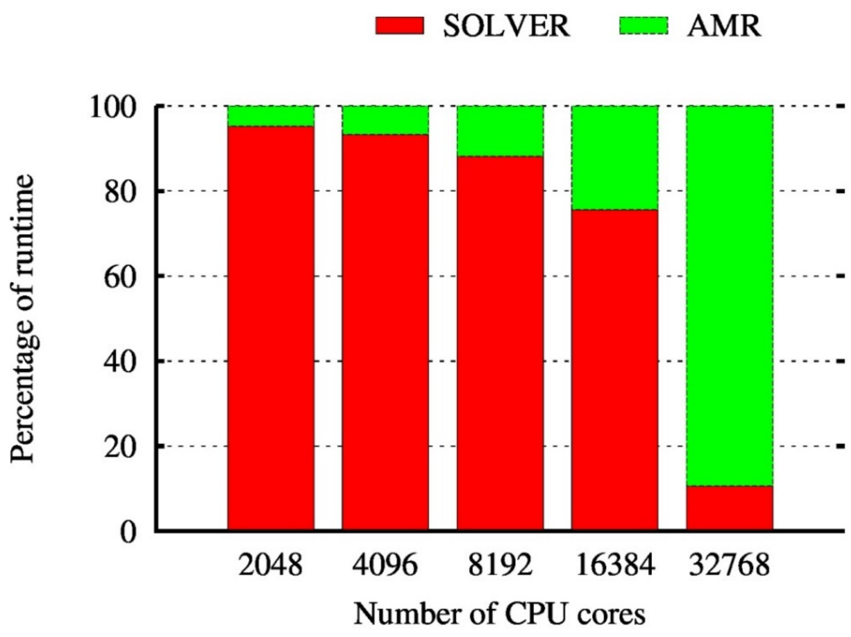


Figure 7.11 Percentage of runtime spent in fluid evolution stage (SOLVER) and mesh adaptation (AMR).

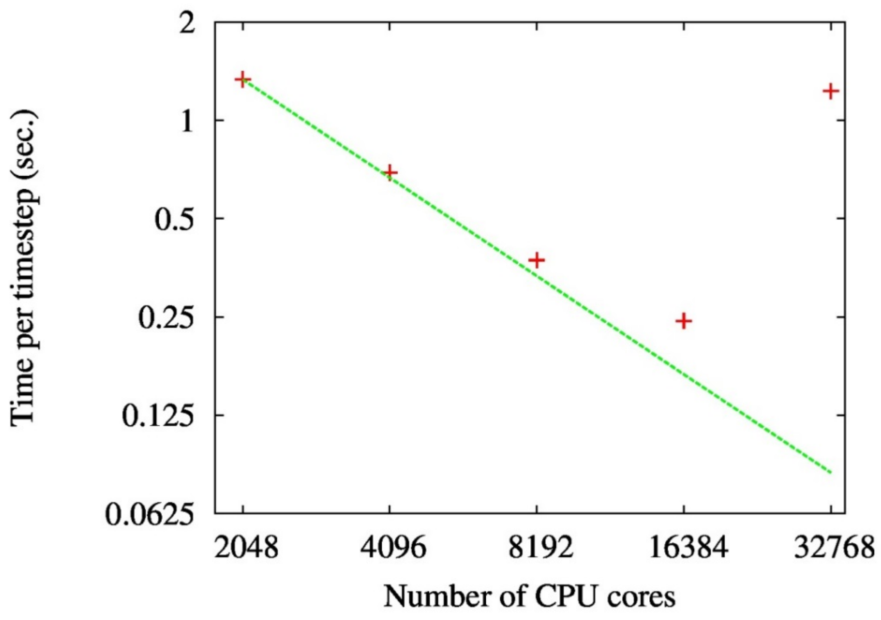


Figure 7.12 Strong scaling of non-conformal Nek5000 with mesh adaptivity for the imbalance tolerance ε=1.01. Green line gives ideal scaling.

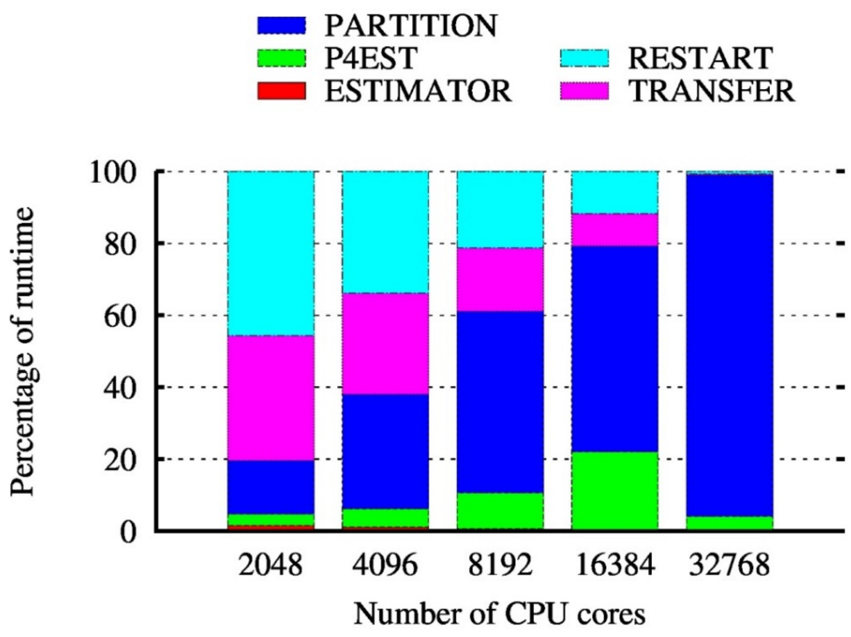


Figure 7.13 Percentage of runtime spent in different stages of mesh adaptation phase as a function of core number.

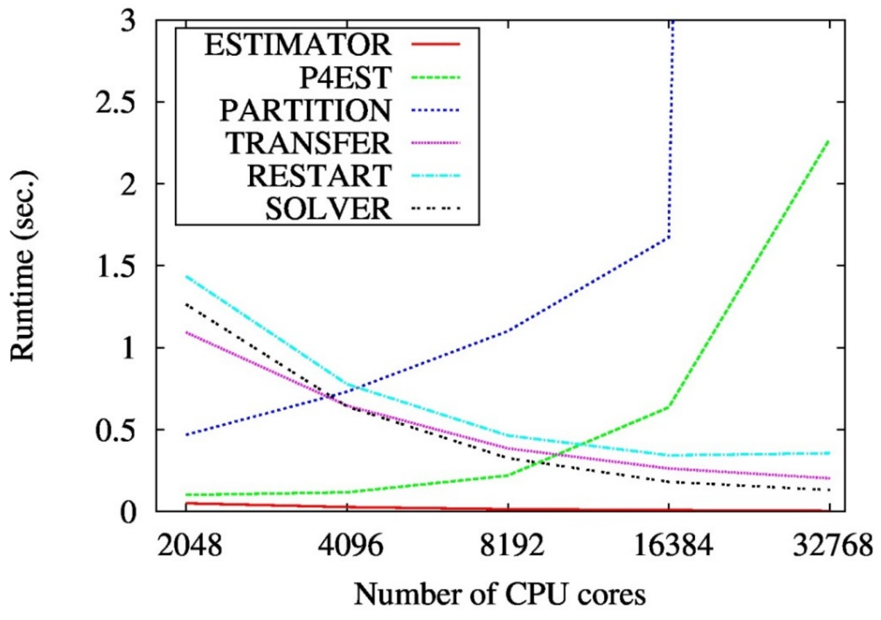


Figure 7.14 Execution time (for single mesh adaptivity cycle) of different tools in AMR stage compared with average time per timestep of Nek5000 solver. In this plot SOLVER does average over the Nek5000 solver stage only and does not include AMR phase.

More insight into the performance of different tools used in the AMR stage give Figure 7.13 showing percentage of runtime spent in different stages of mesh adaptation phase, and Figure 7.14 presenting the execution time of different AMR tools during single mesh adaptivity cycle. They are compared with the average time per timestep of non-conformal Nek5000 solver. Unlike in Figure 7.12, this averaging procedure does not include AMR phase showing the strong scaling of the solver alone. These plots show the error estimator stage (ESTIMATOR) to be negligible, as all the calculations are performed locally. More computationally expensive stages are reset of the Nek5000 solver on the new mesh (RESTART) and sorting/transferring elements between processors (TRANSFER). However, those two phases scale only slightly worse than Nek5000 solver (SOLVER). There is some parallel efficiency decrease in the case of RESTART at NP=32768 caused by communication intensive global numbering of the grid points (performed by p4est) and generation of the new global communicator. The most important limitation is here scalability of used libraries: p4est responsible for mesh regeneration (P4EST) and ParMetis responsible for mesh partitioning (PARTITION). Both perform communication intensive operations (like grid balancing in p4est) and set limit for minimum load, which we estimate to 20 elements per core. However, this value can strongly depend on the polynomial order used for the spectral element method.

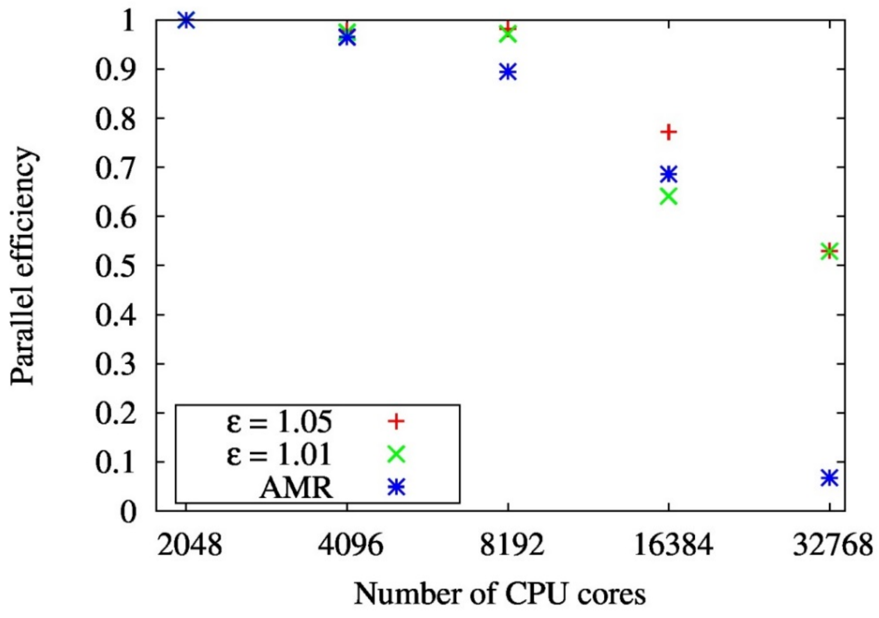


Figure 7.15 Parallel efficiency of the simulations without (ε=1.05; ε=1.01) and with (AMR) grid adaptation for non-conformal version of Nek5000.

We conclude this section with Figure 7.15 presenting parallel efficiency plot of all discussed simulations. It shows comparable parallel efficiency of all the runs with the number of elements per core bigger than 20, and accents importance of the proper balancing of the mesh partitions.

7.4.4 Efficient weak scalability, model problem

GPGPU: Model problem

For the test on Titan, 1024 elements per node and 16th-order polynomials were used for a total of 4,194,304 points per node. Figure 7.16 shows the NekBone weak scaling performance, measured in TFlops, with up to 16,384 GPUs in red color, while the black dashed line represents the ideal weak scaling. The parallel efficiency on 16,384 GPUs was 52.8% compared with single GPU and the maximum performance is 609.8 Tflops.

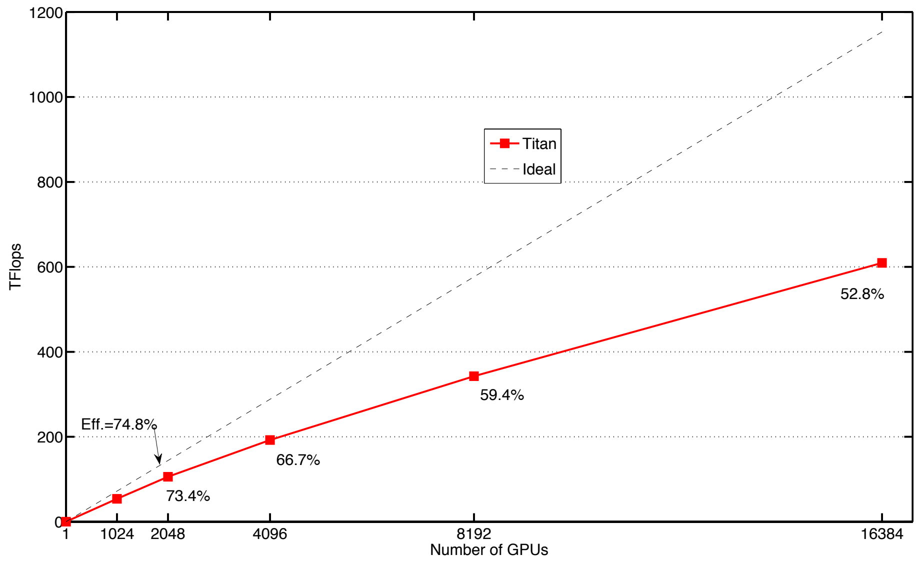


Figure 7.16 Weak scalability results for Nekbone benchmark on the Titan supercomputer with up to 16,384 GPUs (red line) and ideal case (black dashed line).

Comparing the GPU version with the pure CPU version of the NekBone code, a speed-up of 2.4-4.0 times can be obtained, see Figure 7.17.

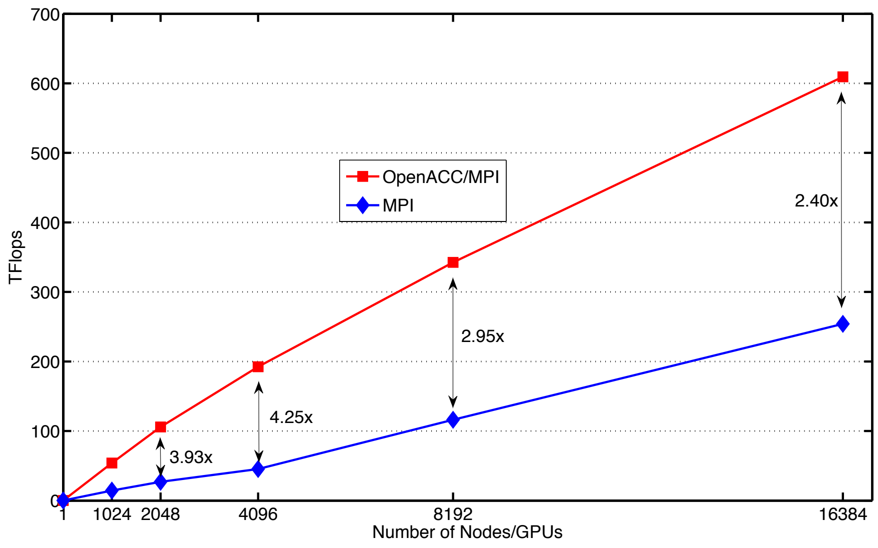


Figure 7.17 Weak scalability results for Nekbone benchmark on the Titan supercomputer with up to 16,384 GPUs compared with full nodes of a pure CPU version.

AMR: Model problem

Presenting weak scalability test we compare, like in the strong scalability case, performance of the non-conformal Nek5000 version without and with grid adaptivity. In the first case simulations are performed with a constant, non-conformal grid partitioned only once during initialization phase. In the second case grid is recreated every 50 solver steps. In all executed runs the grid partitioning is performed by ParMetis and the imbalance tolerance parameter ε is set to 1.01. Adopted test case gives at least 50 elements per core, what allows us to avoid discussed in the previous section problems with limited strong scalability of ParMetis.

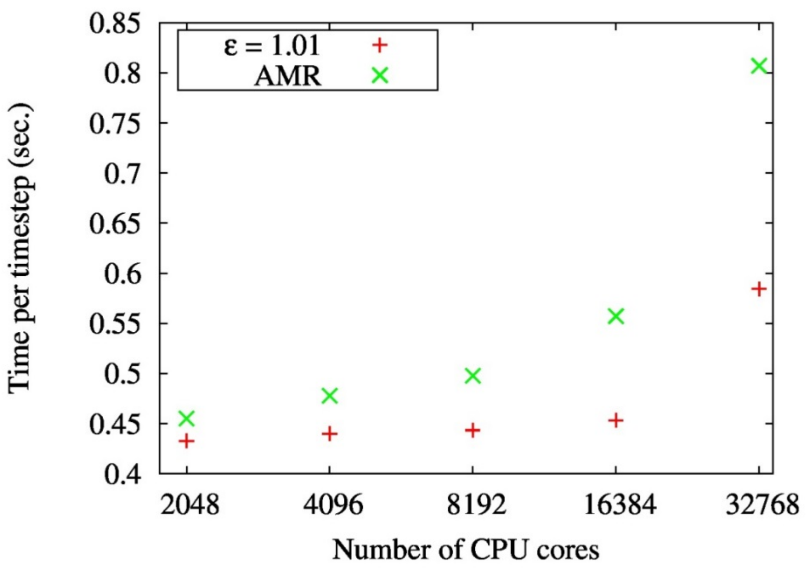


Figure 7.18 Weak scalability of non-conformal Nek5000 without (ε=1.01) and with (AMR) mesh adaptivity. In the AMR case the execution time of the grid adaptation phase is included in the averaging.

Figure 7.18 presents the average time per timestep for the simulations with grid adaptivity turned off (red symbols) and on (green symbols). The element imbalance in both cases is usually smaller than in the strong scalability tests ranging from 4 to 12 elements, but is considerably bigger than element imbalance achieved by native static partitioner of Nek5000. Nek5000 solver itself shows very good parallel performance reaching parallel efficiency of 0.75 for NP=32768 and the global element number equal 1875072.

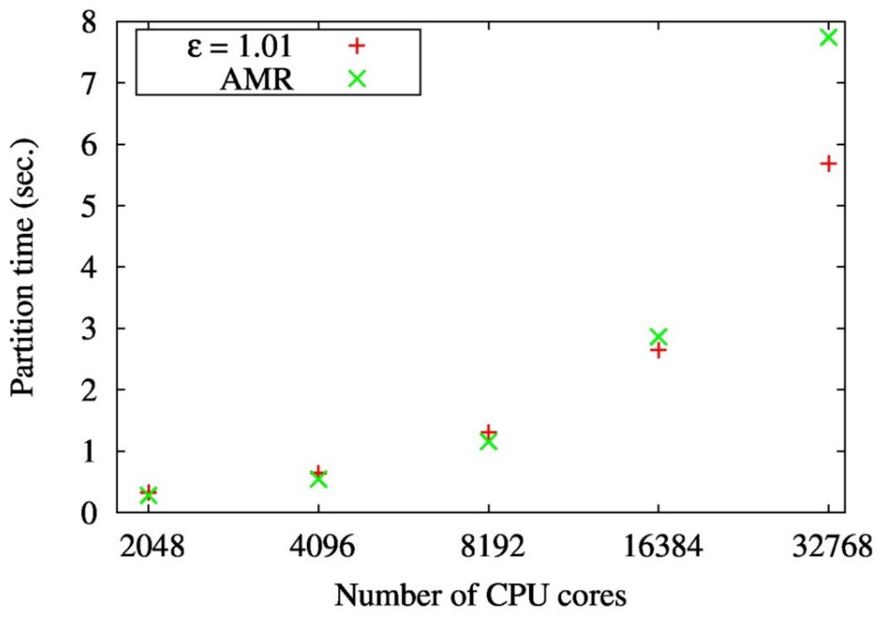


Figure 7.19 24 Execution time for a single grid partitioning as a function of processor number for simulations without (ε=1.01) and with (AMR) mesh adaptivity. In the AMR case plotted time is an average over 80 grid adaptation cycles.

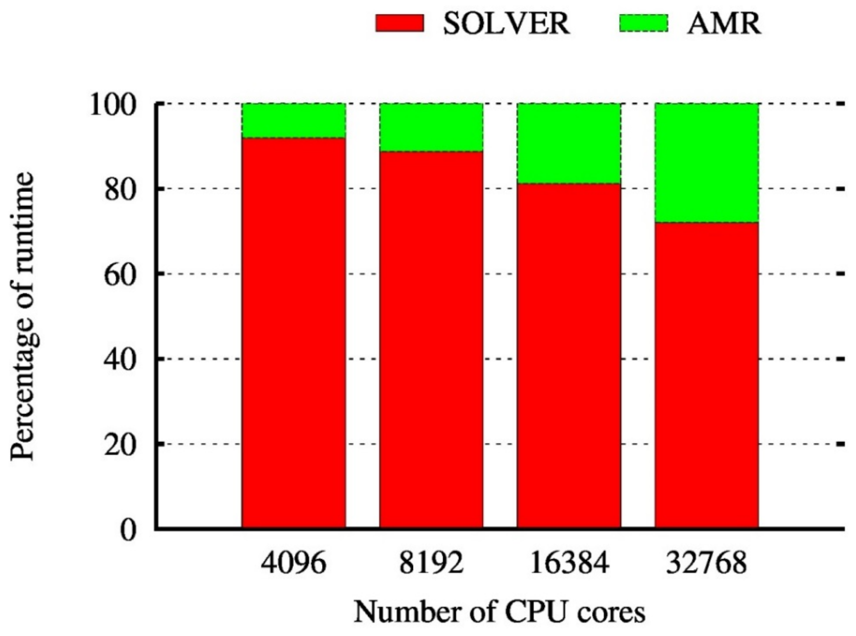


Figure 7.20 Percentage of runtime spent in fluid evolution stage (SOLVER) and mesh adaptation (AMR) for the simulation with grid adaptivity.

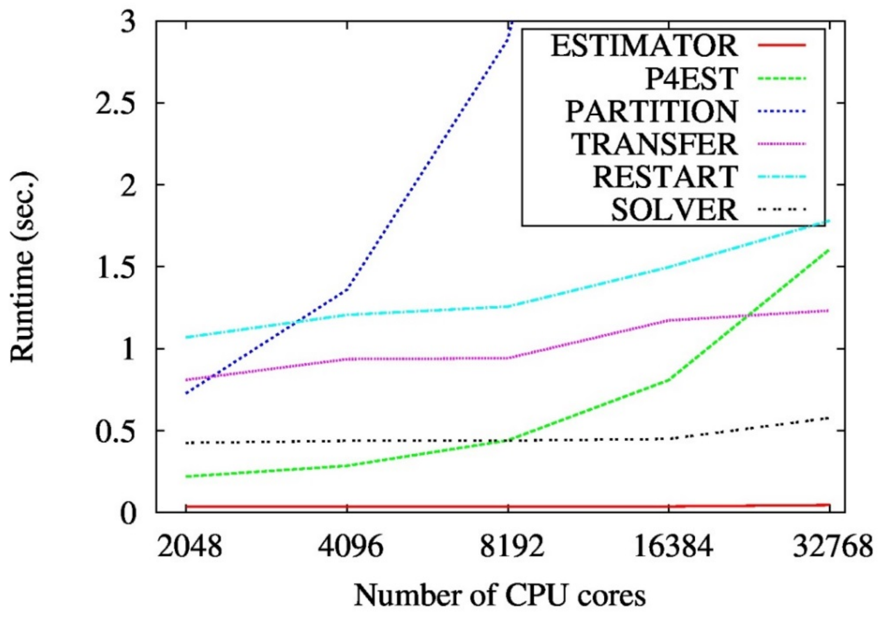


Figure 7.21 Execution time (for single mesh adaptivity cycle) of different tools in AMR stage compared with average time per timestep of Nek5000 solver. In this plot SOLVER does average over the Nek5000 solver stage only and does not include AMR phase.

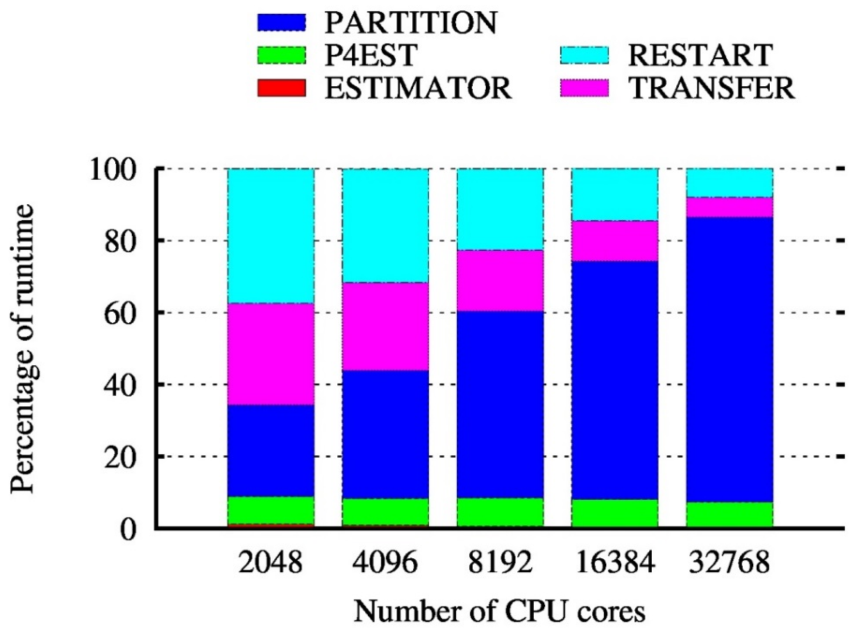


Figure 7.22 Percentage of runtime spent in different stages of mesh adaptation phase as a function of core number.

When the grid adaptivity is included, the parallel efficiency drops to 0.57 showing considerable communication overhead in mesh regeneration stage. It is related to increasing time necessary to perform grid partitioning by ParMetis (see Figure 7.21) however even for the biggest simulation the mesh adaptation phase does not take more than 30% of the runtime (see Figure 7.20). However, Figure 7.21 and Figure 7.22 show grid partitioning to be most costly operation for NP≥4096. Like in the strong scaling case the error estimator (ESTIMATOR) is insignificant for our performance tests. In the similar way we find non-conformal Nek5000 solver to be the most efficiently parallelized component of our code, and the sorting/transfer tool (TRANSFER) scaling reasonably well. There is visible decrease in parallel performance of the Nek5000 restart tool (RESTART) due to communication intensive global numbering of the grid points (performed by p4est) and generation of the new global communicator. On the other hand the scalability of ParMetis is the most important limitation of our implementation requiring some improvements.

We conclude this section with Figure 7.23 presenting parallel efficiency plot of all discussed simulations.

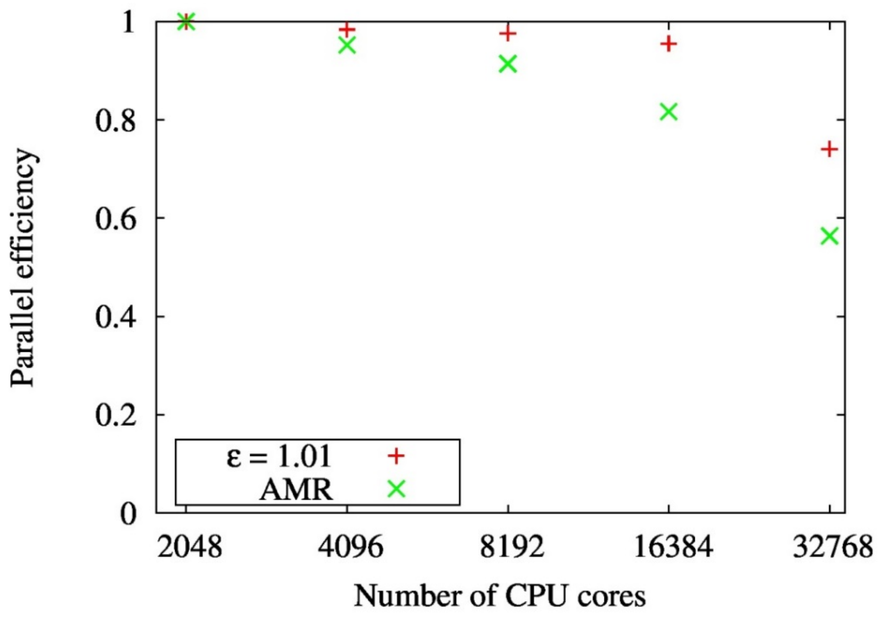


Figure 7.23 Parallel efficiency of the simulations without (ε=1.01) and with (AMR) grid adaptation for non-conformal version of Nek5000.

7.4.5 Efficient strong scalability, exemplar scientific simulation

GPGPU

From Figure 7.2 we know that the performance of kernels (matrix-matrix multiplication) highly depends on the order of polynomial (N) and number of elements (E). Larger values of N give better performance. This may be that the amount of work per thread (which is proportional to N) is greater, which either leads to better kernel efficiency or assists to offset the latency cost of launching kernels. Also the MPI communication overlaps the less workload of GPUs. Consequently, the degradation performance with increase of the number of GPUs for the strong scalability is expected. The performance of OpenACC version is quite different from the original MPI version where parallel efficiency 0.6 has been measured for strong scalability between 32768 and 1048576 MPI ranks, see Figure 7.1.

A Nek5000 simulation of the flow in a straight pipe with 1.2M elements was used to test the performance of the Nek5000 code with OpenACC simulation. The execution time per iteration with different orders of polynomial using the GMRES linear solver and Schwarz preconditioner are compared in

Figure 7.24 and Figure 7.25. The speed-up achieved using OpenACC directives is 1.30 with a 16th order polynomial on 16,384 GPUs compared to 16,384 nodes with 262144 CPU cores. However with a 14th order polynomial we cannot obtain better performance with OpenACC version. Figure 7.2 shows that the performance for the kernels reduces from 61.0 Gflops to 50.5 Gflops with decreasing the number of elements from 512 to 128 per GPU using 14th order of polynomial. larger values of N give better performance. I am not sure of the reason for this, but it might be that the amount of work per thread (which is proportional to N) is greater, which either leads to better kernel efficiency or helps to offset the latency cost of launching kernels. Simulations with larger values of N will run slower than those with smaller values, but larger-N gives better performance.

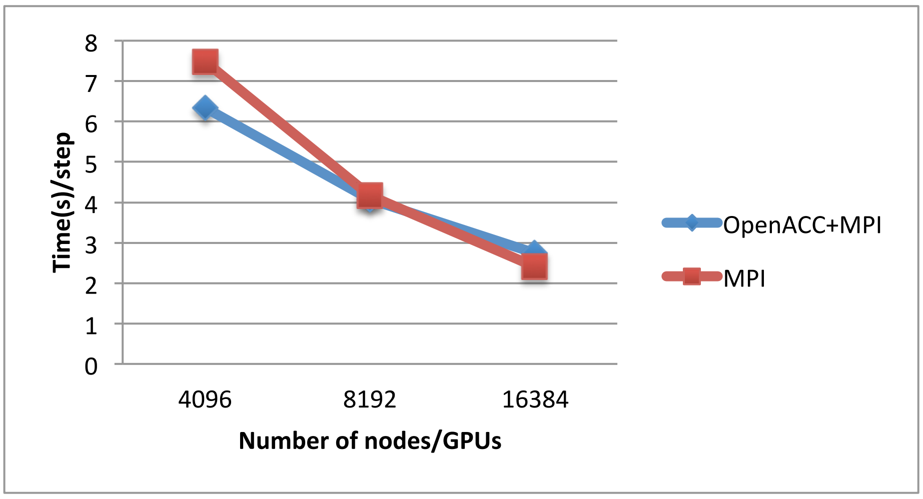


Figure 7.24 The strong scalability for Nek5000 using a 14th order polynomial. The total number or grid points is 3,468 M.

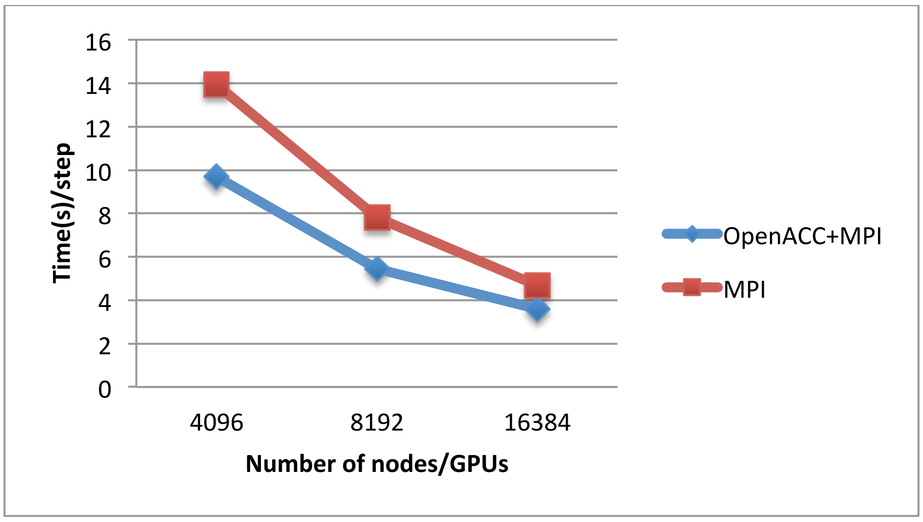


Figure 7.25 The strong scalability for Nek5000 using a 16th order polynomial. The total number of grid points is 5177 M.

7.4.6 Summary of the performance improvements achieved

GPGPU

Nek5000 code has been fully ported to exploit the processing power of multi-GPU systems by using OpenACC compiler directives. The work focused on porting the most time-consuming parts of Nek5000, namely the matrix-matrix multiplication and the preconditioned linear solve operation, to GPGPU. The gather-scatter kernel used with MPI operations was redesigned in order to decrease the amount of data transferred between the host and the accelerator. The speed-up achieved using OpenACC directives is 1.30 with a 16th order polynomial on 16,384 GPUs when compared to 16,384 full CPU nodes having 262,144 CPU cores in total.

We estimate overlapping of the GPU kernels with host-accelerator memory transfers would further increase the performance of the OpenACC version of the Nek5000. Such developments will be part of future research.

AMR

Within CRESTA we implemented in Nek5000 all the tools necessary to dynamically modify the mesh structure during the simulation by changing the global number of elements through h-refinement. It allows for non-conformal meshes which add more flexibility to the grid generation by removing the refinement propagation problem in the conformal meshes. Such propagation would lead to unnecessary elements in the far-field as well as high aspect-ratio elements that are detrimental to the iterative solver performance. It also allows for control of the computational error during the simulation by using proper error estimator.

Depending on the adopted error estimator different goals can be achieved, that is, proper representation of different flow features by resolving advected structures in the flow, or proper treatment of the flow stability properties by resolving sensitive regions in the flow. In our numerical experiments, we found the non-conformal version of Nek5000 to be the most efficiently parallelized component of our code.

Currently the biggest constraint in the parallel scalability is set by the performance of the grid partitioner, showing partitioning from scratch strategy to be inefficient. Right now we are investigating other partitioning strategies. This would unfortunately not allow exascale simulations in which advected flow features are followed either, as the mesh requires continuous adjustment and does not converge to any time independent grid structure. On the other hand, in the stability calculations where the final mesh structure can be time independent, the costly AMR with mesh adaptivity can be used as preprocessing step. Then the non-conformal Nek5000 solver can be used during the main simulation run to have computations at a multi-peta –or exascale.

7.5 References

[1] p4est home page, http://www.p4est.org/

[2] ParMetis home page, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview/

[3] Gottlieb, D. I., and Orszag, S. A. Numerical Analysis of Spectral Methods: Theory and Applications, SIAM-CBMS, Philadelphia, 1977

[4] S. Markidis, J. Gong, M. Schliephake, E. Laure, A. Hart, D. Henty, K. Heisey and P. F. Fischer, OpenACC Acceleration of Nek5000, Spectral Element Code Advances in Engineering Software Journal (under review)

[5] J. Gong, S. Markidis, M. Schliephake, E. Laure, D.S. Henningson, P. Schlatter, A. Peplinski, A. Hart, J. Doleschal, D. Henty and P. F. Fischer, Nek5000 with OpenACC, Lecture Notes in Computer Science, Springer (accepted)

[6] P. Schlatter and G. K. El Khoury, Turbulent flow in pipes, PDC newsletter, 2012 No.1

[7] Nek5000 web page, Available online at: http://nek5000.mcs.anl.gov

[8] J. Malm, P. Schlatter and D. S. Henningson, *“Coherent structures and dominant frequencies in a turbulent three-dimensional diffuser”*, J. Fluid Mech. 2012.

[9] Bernd Mohr, Wolfgang Frings, “Extreme Scaling Workshop 2010 Report”, Jülich Supercomputing Centre, 2010. Available online at: http://juser.fz-juelich.de/record/9600/files/ib-2010-03.pdf.

[10] Paul Fischer, “Nek5000 Tutorial”, 2010. Available online at: http://www.mcs.anl.gov/~fischer/nek5000/fischer_nek5000_dec2010.pdf.

[11] Nekbone benchmark web page, available online at: https://cesar.mcs.anl.gov/content/software/thermal_hydraulics

8 OpenFOAM

OpenFOAM® is an open source library for computational multiphysics and especially computational fluid dynamics (CFD) problems. The library is a "toolbox" which provides a selection of different solvers as well as routines for various kinds of analysis, pre- and post-processing. OpenFOAM® is licensed under the GPL. As such, modifications have been made to the code by different parties at different times and several versions are in common use. In this project, we consider the official release from the OpenFOAM® foundation (a not-for profit organization, wholly owned by OpenCFD Ltd.), and the release from the OpenFOAM® Extend project.

8.1 Summary of the previous roadmaps

Task	Scheduled date	Status
Benchmarking of the latest version of the code	M36	Completed
Code analysis of latest version of code	M36	Completed
Performance analysis of kernels, libraries	M36	Completed
Test case 02: Pump turbine power plant with OpenFOAM-2.1	M18	Completed
Scientific results for the pump turbine flow simulation	M36	Completed
Iterative performance improvement	M36	Cancelled

Table 8.1 Summary of the previous roadmaps for OpenFOAM

Benchmarking of the latest version of the code

Detailed benchmarking of the code proved considerably more difficult than what was initially assumed as a number of tools, such as CrayPAT, failed to instrument the OpenFOAM binary. Benchmarking was concluded by collaborating with Vampir design team and instrumenting parts of the program manually.

Code analysis of latest version of code

The task has been completed, resulting in an analysis of the structure of OpenFOAM. For a detailed analysis see Section 8.2.

Performance analysis of kernels, libraries

Performance analysis has been concluded on Hornet (Cray XC40) by using the Vampir tool with analysis given in Section 8.2.

Test case 02: Pump turbine power plant with OpenFOAM-2.1

The test case was run on Hermit. As the parallel scalability was poor, the focus is on the scientific results with version 1.6-ext.

Scientific results for the pump turbine flow simulation

Turbulence modelling is a key point in the simulation of pump turbines. In certain operating conditions strong swirling flows occurs in the draft tube. This leads to strong pressure pulsation that in turn might lead to structural damage. The investigation was done with version 1.6-ext.

Iterative performance improvement

When performing the benchmarking and profiling tasks, it became evident that with the resources available, OpenFOAM cannot be modified to be an exascale code and this task was cancelled. A more detailed analysis is given in Section 8.2.

8.2 Roadmap to exascale

OpenFOAM, as it is currently constructed, is not an exascale code, at least for the exemplar scientific use cases available within CRESTA. In this section, we give an overview of the less than ideal scalability of the code followed by an analysis and recommendations for future improvements of the code.

8.2.1 Efficient strong scalability: model problem

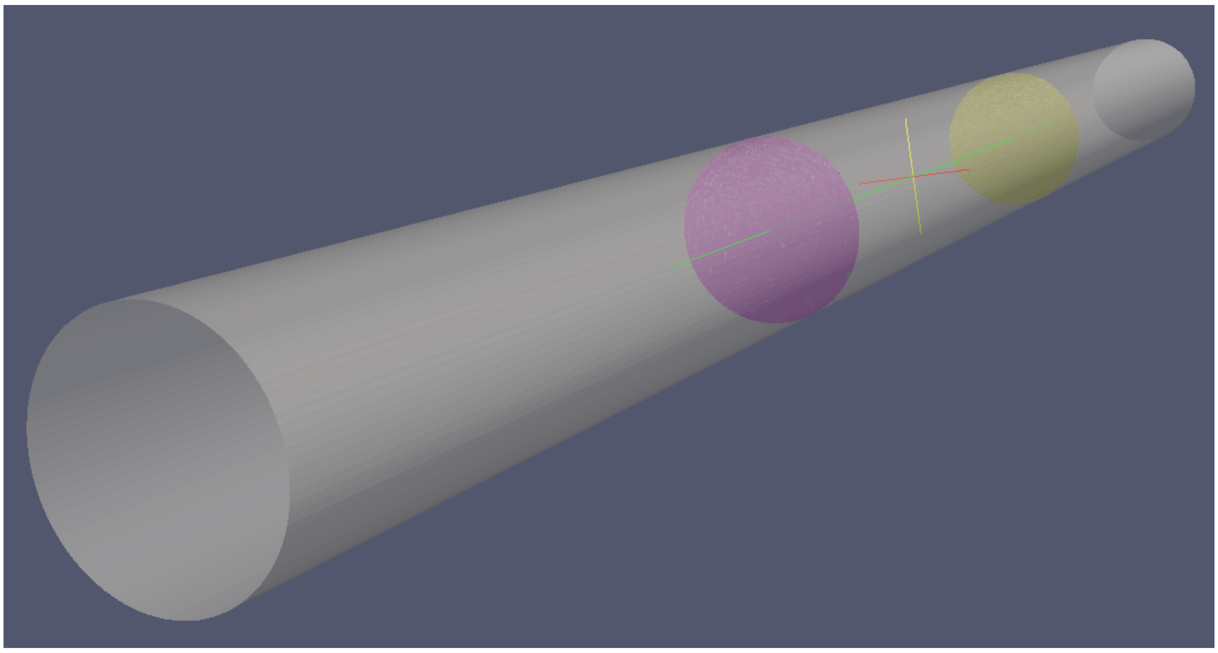


Figure 8.1: Pipe test case with two GGI interfaces (OpenFOAM-1.6-ext).

8.2.2 The pipe test case: introduction pipe test case (OpenFOAM-1.6-ext)

As a test case for a flow involving rotating meshes, we use a flow through a pipe. The pipe is divided into three cylinders and the cylinder in the middle is set to rotate about its axis. Boundaries of the pipes sections are set without any velocity boundary conditions and therefore the rotation of the pipe midsection has no effect on the properties of the flow. Coupling of meshes is implemented via two GGI interfaces, sketched in the Figure 8.1 as the purple and yellow areas of a circle. The advantage of this test case is that it uses GGI without the complexity of a whole hydraulic machine.

The pipe is discretized into 6940350 cells, and its domain is decomposed by the OpenFOAM utility "decomposePar" with the "scotch" method enabled. Inlet and outlet boundaries are defined at the respective ends of the pipe, and the initial conditions are set accordingly. For the numerical solution, the pimple method is used with the rotating mesh approach, thus the OpenFOAM solver "**pimpleDyMFoam**" is chosen.

8.2.3 Strong scaling results

Figure 8.2 shows the results of a scaling test of the pipe test case on the HPC system Hornet at HLRS. One node on Hornet consists of 24 cores, so the range of parallelism tested is between one node and 64 nodes, or from 24 cores to 1536 cores. Between one node and 16 nodes we can see a nearly ideal speedup, at some points even super linear.

As a rule of thumb, CFD problems using FVM for its solution should be stated with at least 50.000 cells per core in order to get a decent speedup. As we can see in Figure 8.2, OpenFOAM shows good scaling up to 16 nodes, or 384 cores, where we are using at about 18.000 cells per core. Therefore, in this test case OpenFOAM shows quite a good scaling behaviour for a CFD code. Nevertheless, as the number of cells per core would need to be significantly less than this for OpenFOAM to reach exascale.

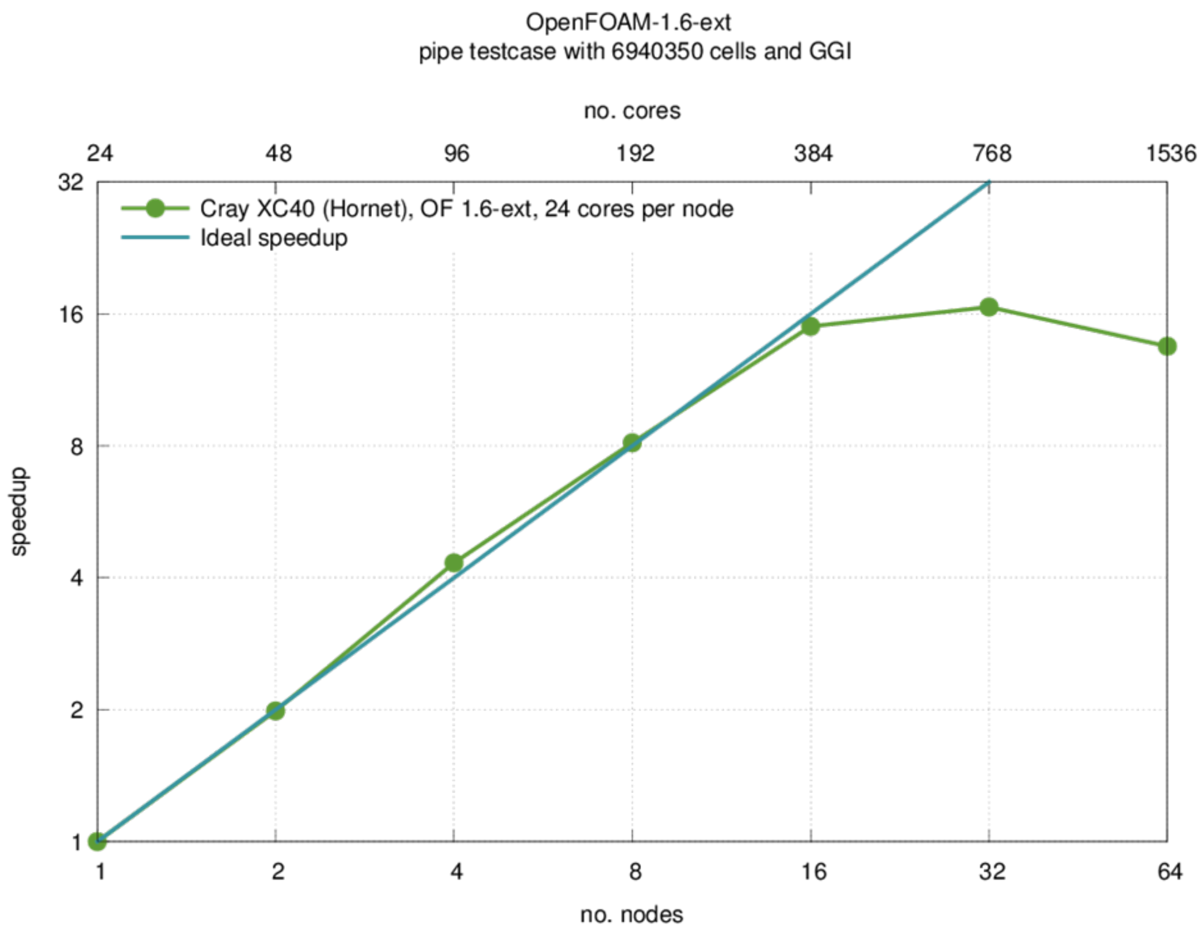


Figure 8.2: Scaling test on the pipe test case with about 7 million cells.

8.2.4 Instrumentation of OpenFOAM-1.6-extend with Vampir

After numerous tries, we finally succeeded in instrumenting OpenFOAM with Vampir for tracing experiments. In this section, we would like to give a short overview on how the instrumentation process was performed.

First we set the compiler command environment variable to the Vampir wrapper as follows:

```
CC  = vtcxx -vt:inst manual -vt:cxx CC -vt:mpi -DVTRACE
```

Notice that we are used manual instrumentation user functions as well as instrumentation of MPI calls.

We then compiled OpenFOAM in a standard way, i.e.,

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_DIR=path_to_install_to
make
```

Automatic instrumentation of OpenFOAM was feasible because OpenFOAM uses a lot of calls to small templated functions. Normally, these functions would be inlined and the overall context optimised by the compiler. This is not the case when using automatic instrumentation performed by a tool such as Vampir, since instrumenting function calls prevents the compiler from inlining the functions and optimizing their context.

As an example of instrumentation preventing optimizations, consider the implementation of a basic vector operation in the file "`src/foam/primitives/VectorSpace/VectorSpaceM.H`":

```
template<int N, int I>
class VectorSpaceOps
{
```

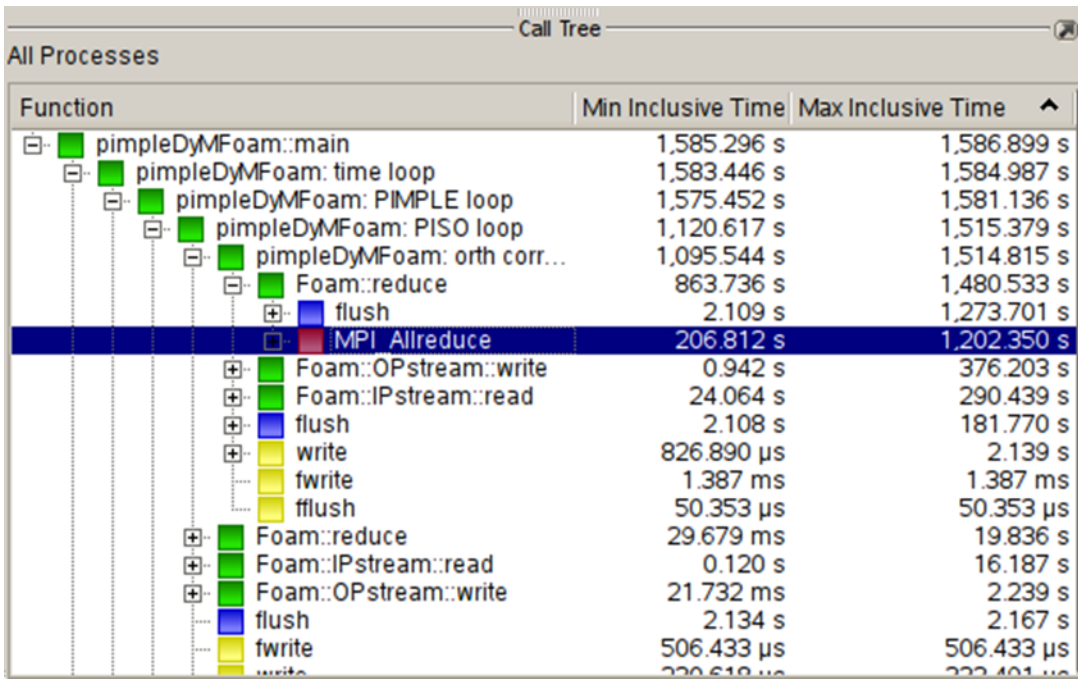
```
public:
    static const int endLoop = (I < N-1) ? 1 : 0;
    ...
    template<class V1, class V2, class EqOp>
    static inline void eqOp(V1& vs1, const V2& vs2,
EqOp eo)
    {
        eo(vs1.v_[I], vs2.v_[I]);
        VectorSpaceOps<endLoop*N,
endLoop*(I+1)>::eqOp(vs1, vs2, eo);
    }
    ...
};
```

In the code excerpt, an iterative call to the scalar function “`eo`” in a vector operation is implemented by a recursive call to the member function “`eqOp`”, which sits in the template class “`VectorSpaceOps`”. The iterations are controlled by a template parameter “`int I`”, which operates as a compile time constant loop iterator. In general, such constructs will be inlined by modern optimizing compilers. If every call to the member function “`eqOp`” and the basic operator function “`eo`” would be instrumented, the tracing of these functions would cost more than performing the actual scalar operations within the vector operation, strongly distorting the performance measurements.

8.2.5 Tracing of pimpleDyMFoam on pipe test case with Vampir

As an example of using Vampir on OpenFOAM for tracing experiments, consider the **pimpleDyMFoam** solver and a smaller version of the pipe test case in terms of cell numbers (866295 cells). We manually instrumented the **pimpleDyMFoam** for tracing every entry in a loop of the solver implementation. Figure 8.3 shows the visual call tree output of Vampir. The tracing was conducted with a number of processors with which the parallel efficiency was known to have been decreased, i.e., no further speedup from increasing the number of processors had been acquired.

As can be seen from the trace in Figure 8.3, most of the time is being spent in the MPI_Allreduce call within the inner-most loop. The MPI_Allreduce in question belongs to the dot-product (scalarproduct) of the PCG-solver, which is used in this test case for solving the pressure equation. Because the PCG method for solving the pressure equation is not a special feature of OpenFOAM, we argue that a review of the applied algorithms is needed here to evolve OpenFOAM towards exascale.



Function	Min Inclusive Time	Max Inclusive Time
🟩 pimpleDyMFoam::main	1,585.296 s	1,586.899 s
🟩 pimpleDyMFoam: time loop	1,583.446 s	1,584.987 s
🟩 pimpleDyMFoam: PIMPLE loop	1,575.452 s	1,581.136 s
🟩 pimpleDyMFoam: PISO loop	1,120.617 s	1,515.379 s
🟩 pimpleDyMFoam: orth corr...	1,095.544 s	1,514.815 s
🟩 Foam::reduce	863.736 s	1,480.533 s
🟦 flush	2.109 s	1,273.701 s
🟥 MPI_Allreduce	206.812 s	1,202.350 s
🟩 Foam::OPstream::write	0.942 s	376.203 s
🟩 Foam::IPstream::read	24.064 s	290.439 s
🟦 flush	2.108 s	181.770 s
🟨 write	826.890 µs	2.139 s
🟨 fwrite	1.387 ms	1.387 ms
🟨 fflush	50.353 µs	50.353 µs
🟩 Foam::reduce	29.679 ms	19.836 s
🟩 Foam::IPstream::read	0.120 s	16.187 s
🟩 Foam::OPstream::write	21.732 ms	2.239 s
🟦 flush	2.134 s	2.167 s
🟨 fwrite	506.433 µs	506.433 µs

Figure 8.3: Vampir trace call tree of manually instrumented pimpleDyMFoam on pipe test case.

8.2.6 Roadmap to exascale: OpenFOAM conclusions

Updating rotating meshes as described in Section 8.2.2 involves changes in the MPI communicator, i.e. in the communication pattern used between the computing cores. Unfortunately, the typical use case for MPI communicators is that they are set up once in the beginning of a simulation and not modified during runtime as modifications require a collective operation within the parent communicator. Our measurements indicate that the time spent in the mesh update remains nearly constant between 256 and 1024 cores, i.e., it does not parallelize at all.

Besides the rotating mesh interface, there are other well-known problems limiting the scalability of all CFD codes, not only OpenFOAM. Solving the coupled pressure and velocity equations, for example, is generally a problem in simulating incompressible fluids. In addition, the widely-used linear solver CG (conjugate gradients) and derived solvers require the frequent calculation of global scalar products which act as a global barriers (for analysis see the previous subsection). In order to perform exascale computations with CFD codes, fundamental research on new mathematical algorithms has to be performed.

A problem specific to OpenFOAM is its complexity. For instance, an attempt to implement parallel I/O by using MPI I/O in OpenFOAM failed, since MPI experts within CRESTA, could not get the needed insight into the code due to the complexity of the codebase and the resource limitations of the project.

OpenFOAM takes advantage of the possibilities of C++, such as expression templates, operator overloading and complex class hierarchies. This is a design decision by the developers of OpenFOAM, as their goal has been to create a widely-applicable CFD code where the users can easily implement physical models without taking care of matrix assembly and linear solver technology. Such features are highly appreciated by users, but makes any changes to modify the core structure very time consuming for developers who are not familiar with the design. In order to be prepared to exploit the computing capabilities of future exascale systems, we strongly recommend OpenFOAM, or another CDF code with similar capabilities, undergo disruptive changes or a partial rewrite under a close collaboration of a group of experts in HPC, numerics and CFD.

8.3 Exemplar scientific simulation

Since the parallel scalability of OpenFOAM is not sufficient for exascale, we not focus solely the exemplar scientific simulation of OpenFOAM, i.e., pump turbine test case. For details and further description of the test case, refer to the CRESTA Deliverable D6.4 [6].

8.3.1 Choosing an OpenFOAM version: OpenFOAM 1.6-ext versus OpenFOAM 2.1.1

The pump turbine flow simulation was tested with the newer version of OpenFOAM, namely version 2.1.1, where the coupling interface between the (rotating) meshes is called AMI (arbitrary mesh interface). As an example case, a mesh with 40 million elements was chosen. A speedup test was done between 64 and 1024 cores (see Figure 8.4). Beyond 128 cores the performance of version 2.1.1 strongly and rapidly decreases. Version 1.6-ext with GGI interface scales quite well up to 1024 cores instead. For this reason we further concentrated and used version 1.6-ext for the pump turbine application.

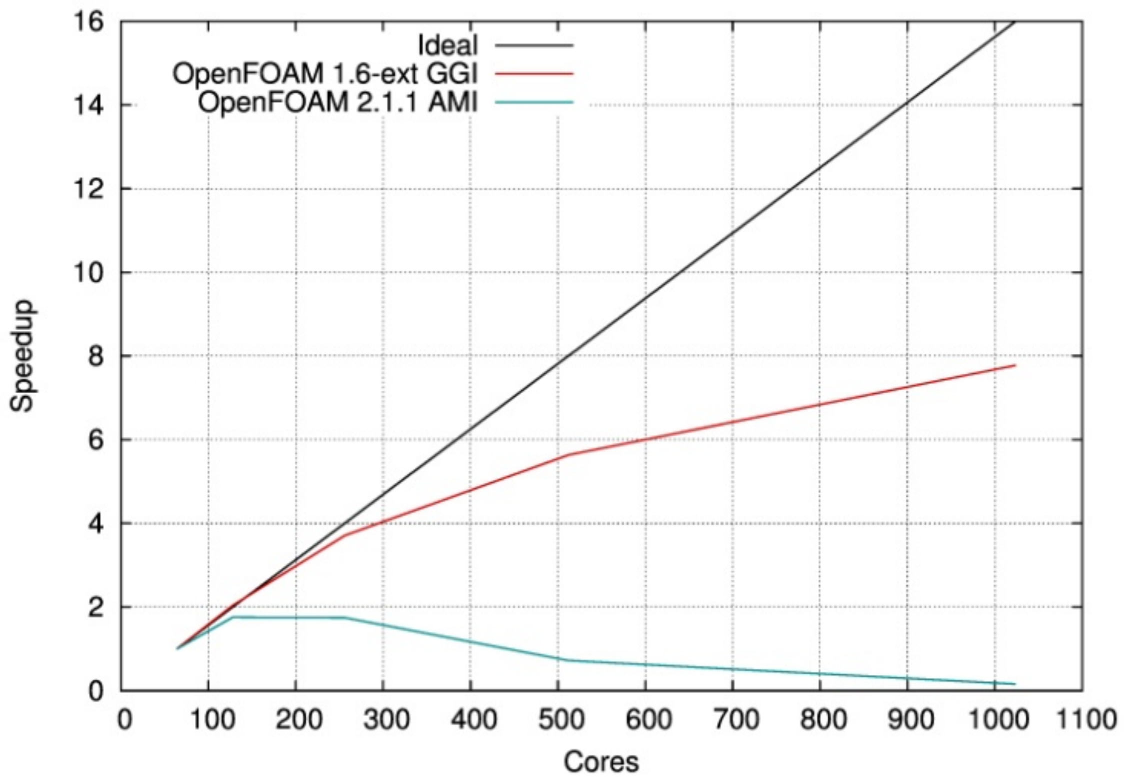


Figure 8.4: Parallel Performance of OpenFOAM version 1.6-ext and 2.1.1 for the pump turbine simulation

8.3.2 Scientific results

For a pump turbine flow simulation it is not only sufficient to resolve the large flow structures, but also small turbulent scales. As standard RANS models only predict large turbulent scales, an eddy resolving method – LES – must be applied. With LES it is computationally extreme expensive to resolve the small eddies in the boundary layer. For this reason we implemented the IDDES (improved delayed detached eddy simulation) [5] type turbulence model based on the RANS-SST model [4]. This model uses RANS in the boundary layer and LES in the core flow using some blending functions.

A pure LES would require some billions of elements for the complete pump turbine. The problem is that the pump turbine mainly consists of walls: the spiral casing, 20 stay and 20 guide vanes, 7 runner blades and the draft tube with extension and bending (see Figure 8.5). A rather coarse RANS mesh would require around 10 million elements (10M). We investigated this mesh and a refinement with 20 million elements

(20M). This mesh is still away from ideal. An ideal mesh would require at least 100 million elements for the pump turbine.

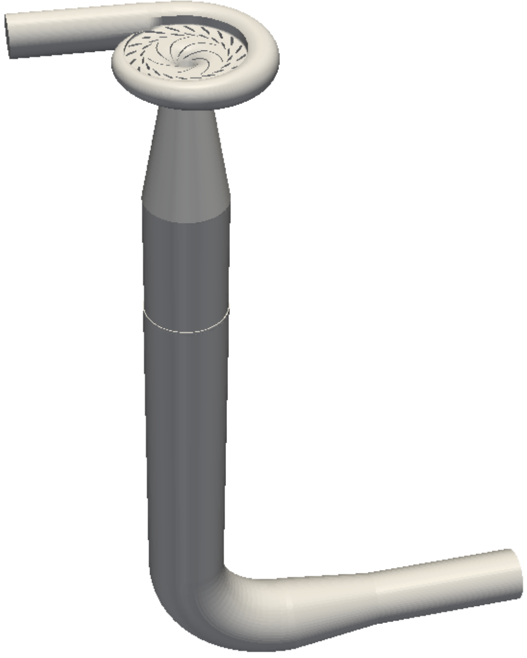


Figure 8.5: Geometry of the pump turbine used for the flow simulations

Beneath the large mesh size, quite small time steps of 0.1ms are needed. That means around 400 time steps per runner revolution. To get valuable statistics of turbulent data 50 runner revolutions are necessary, leading to 20000 time steps for the 20M mesh. Larger meshes would require smaller time steps to keep the Courant number below one in the region of resolved turbulence.

The simulations were done with OpenFOAM-1.6-ext on Hector XE6 and Hermit XE6 on 256 cores for the 10M mesh and 512 cores for the 20M mesh. The simulation time for the coarse mesh is 90 hours and for the fine mesh 110 hours.

The results for the pump turbine flow simulation were published at the 10th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements [2] and at the High Performance Computing in Science and Engineering '14 [3].

The axial velocity distribution in a cutting plane in the draft tube is depicted in Figure 8.6. It is quite obvious that the RANS model only resolves large flow structures and no small turbulent scales (left). The IDDES-type turbulence model is resolving small turbulence structures (middle and right).

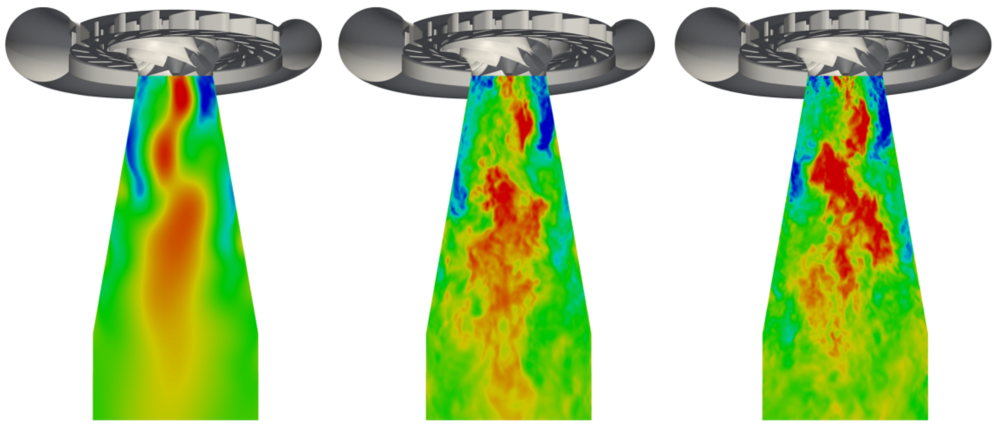


Figure 8.6: Axial velocity distribution in a cutting plane in the draft tube; colour range: -5 to 2 m/s; left and middle 10M mesh, right 20M mesh; left RANS-SST model, middle and right IDDES-SST model

A comparison of the turbulent flow simulation in the draft tube cone with measurements from [1] is depicted in Figure 8.7. The RANS simulation significantly overestimates the backflow in the core region. With the usage of the hybrid IDDES turbulence model, the velocity distribution fits better with the experimental data.

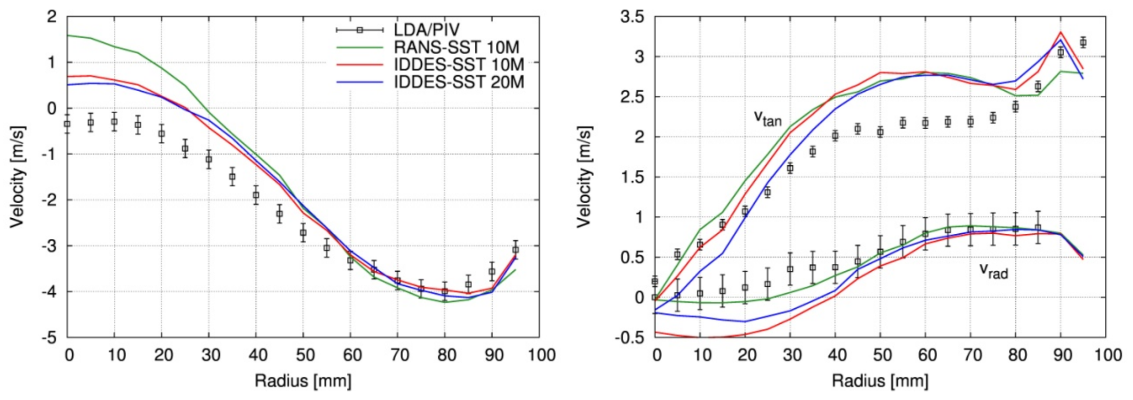


Figure 8.7: Velocity distribution in the draft tube for the axial (left) and the tangential and radial component (right) in comparison with measurements

A consideration of the turbulent content of the flow is done for the turbulent kinetic energy (see Figure 8.8). The RANS simulation clearly underestimates the amount of turbulent kinetic energy (TKE) in the core flow with and without the modeled content of the turbulence model. The content of the modeled TKE is quite small for the IDDES simulations. The coarse mesh IDDES simulation shows too high values of TKE in the core. The results obtained by the IDDES-SST 20M simulation agree well with the measurements instead across the whole diameter.

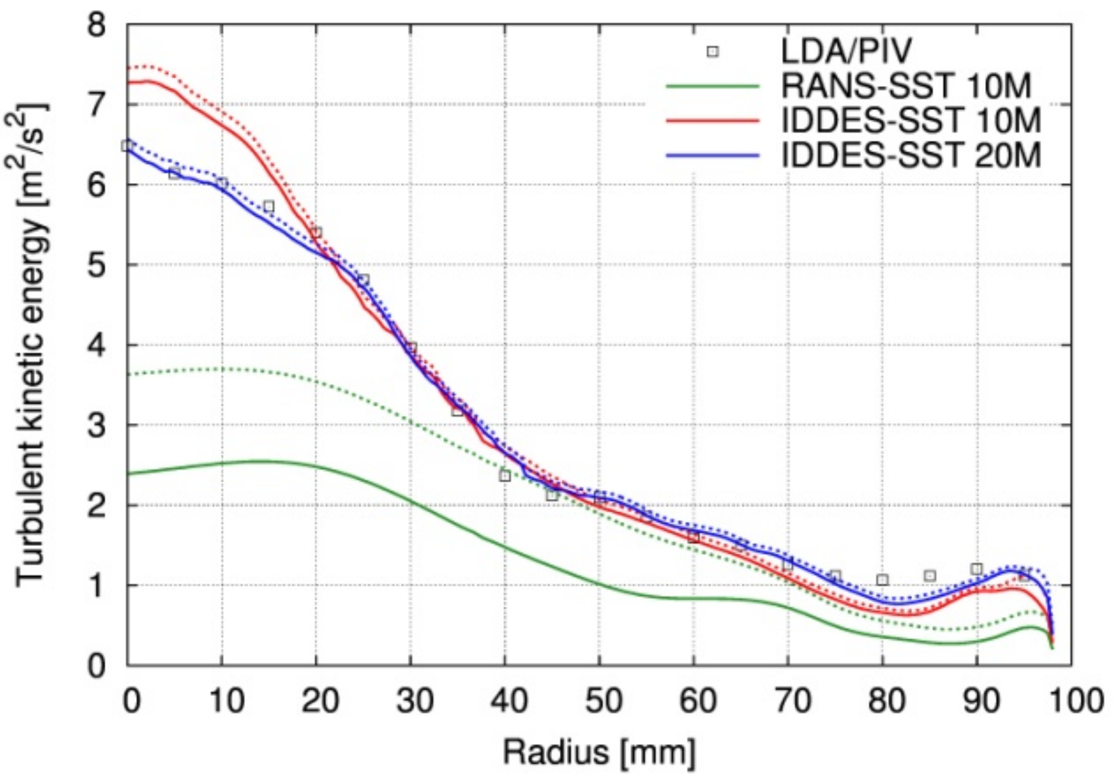


Figure 8.8: Turbulent kinetic energy distribution in the draft tube in comparison with measurements; solid: resolved part, dashed: resolved and modeled part

Due to the strong tangential and axial velocity component for the outer radius, in the core a low pressure zone arises, the so called vortex rope phenomenon. The IDDES-SST model resolves a larger vortex rope (see Figure 8.9), as the RANS model is too dissipative. The hybrid model is able to resolve small structures at the end of the vortex rope. The results also show that the finer the mesh for the hybrid model the finer the turbulent structures are. At the end of the vortex rope it decays to small turbulent structures.

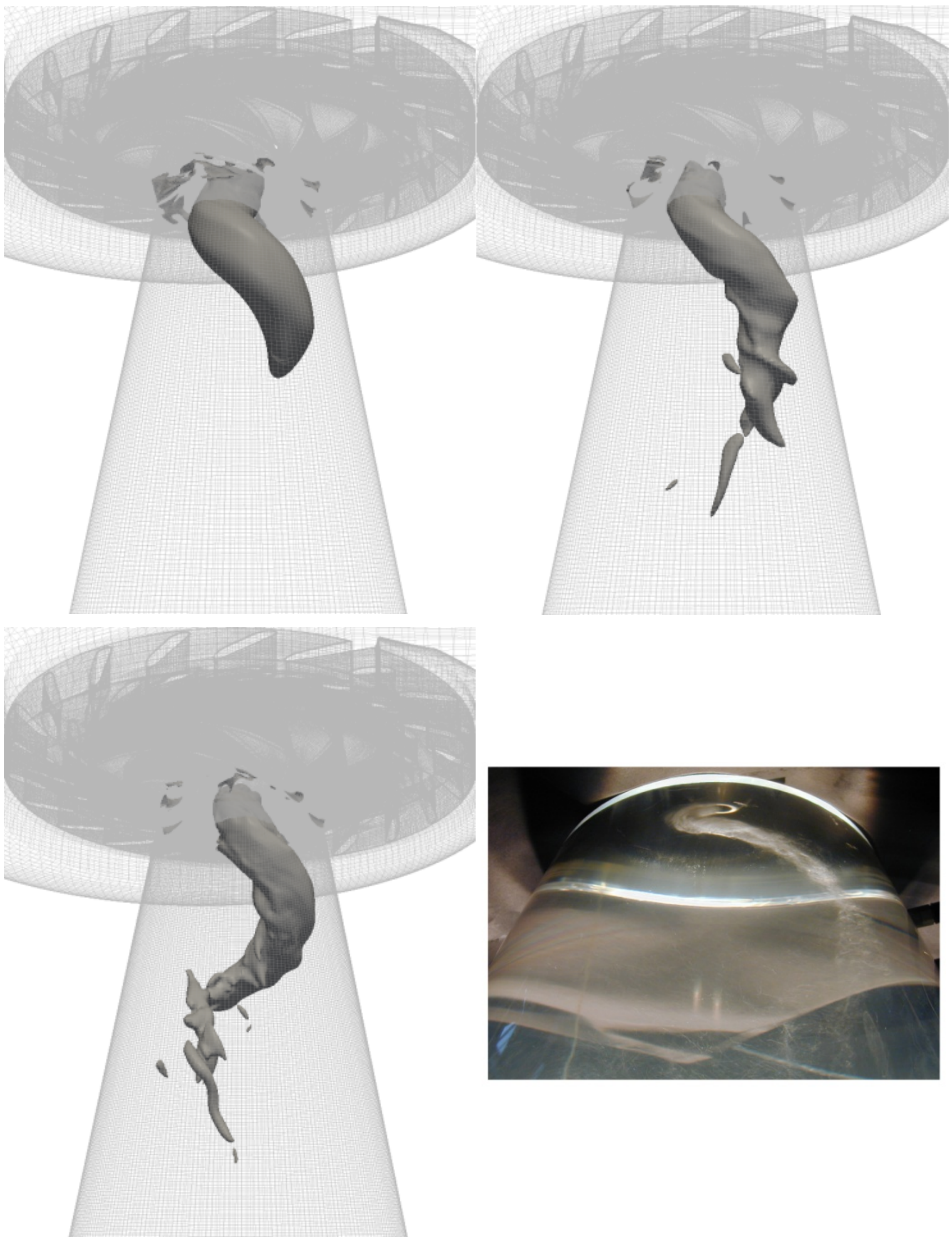


Figure 8.9: Visualization of the vortex rope phenomenon with iso-surface of pressure; top: 10M mesh, left: RANS-SST, right: IDDES-SST, bottom: left: 20M mesh IDDES-SST, right: experiment

The smaller vortex rope obtained by the RANS-SST simulation leads to a higher frequency for the pressure fluctuation compared to experimental results (see Figure 8.10, left). The coarse mesh IDDES simulation overestimates the pressure amplitude and somewhat the frequency. The fine mesh IDDES simulation fits the measurements best for the frequency predicting slightly higher pressure amplitude. Generally, the higher harmonics of the vortex rope frequency are better represented by the IDDES simulations than the RANS simulations. This trend is also visible for the frequency generated by the runner blade wakes (see Figure 8.10, right). Using the RANS turbulence model they are not able to be simulated in contrast to using the IDDES turbulence model.

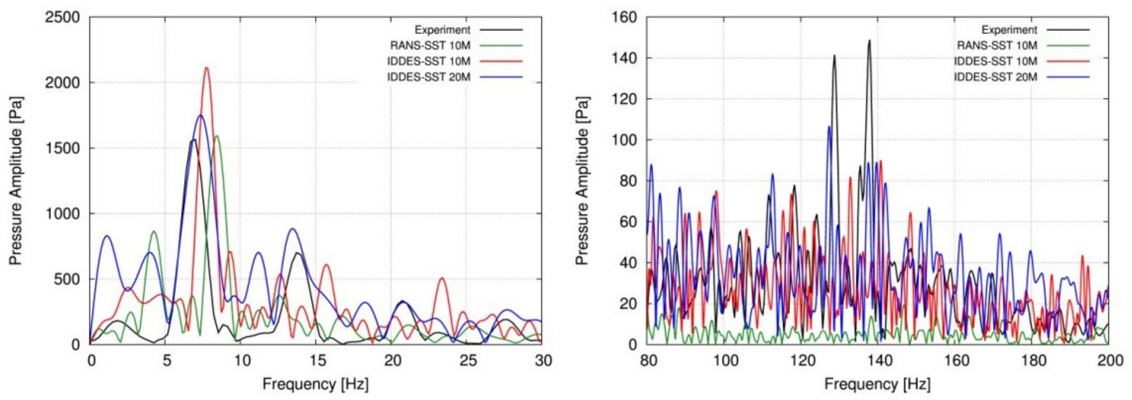


Figure 8.10 Pressure pulsations in the draft tube cone for different simulations and experiment

Conclusions

The IDDES-type turbulence model was successfully applied to a Francis pump turbine flow simulation at turbine part load operating conditions. The validation against measurements shows a better representation of the velocity field in the region where the vortex rope occurs compared to RANS simulations. The vortex rope phenomenon itself can be better resolved using the hybrid turbulence model. The resulting pressure fluctuations, validated against measurements as well, show a better resolution of the vortex rope induced frequencies, if the mesh resolution is fine enough. Furthermore, only the IDDES turbulence model is able to resolve details like pressure pulsation generated by the runner blade wakes. The IDDES turbulence model predicts the turbulent kinetic energy very well compared to measurements.

The used mesh sizes are at the lower limit of what has to be used for the hybrid RANS-LES turbulence model. Resolving the boundary layer and the core flow region requires much more elements, namely at least 100 million elements. Despite of the large number of elements, this is still far away from a well resolved LES resolution. This means that in the future more computational resources are needed. It seems to be quite challenging to optimize such a CFD code for this kind of application.

8.4 References

[1] Kirschner O.: Experimentelle Untersuchung des Wirbelzopfes im geraden Saugrohr einer Modell-Pumpturbine, Dissertation, IHS-Mitteilungen, Vol. 32., 2011

[2] Krappel T., Kuhlmann H., Kirschner O., Ruprecht A., Riedelbauch S.: Validation of an IDDES-type Turbulence Model and Application to a Francis Pump Turbine Flow Simulation, 10th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements, Marbella, Spain, 2014

[3] Krappel T., Ruprecht A., Riedelbauch S.: Flow Simulation of Francis Turbines using Hybrid RANS-LES Turbulence Models, High Performance Computing in Science and Engineering '14, Springer 2014 (to be published)

[4] Menter F., Kuntz M., Langtry R.: Ten Years of Industrial Experience with the SST Turbulence Model, In Turbulence, Heat and Mass Transfer, Vol. 4, pp.625-632, 2003

[5] Shur, M.L., Spalart, P.R., Strelets, M.K., Travin, A.K.: A hybrid RANS-LES approach with delayed-DES and wall-modeled LES capabilities, International Journal of Heat and Fluid Flow 29, pp. 1638-1649, 2008

[6] Exemplar scientific simulations, CRESTA Deliverable D6.4.