D6.1.2 – Roadmap to exascale (update 1)

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 Exascale computing, software and simulation

Due date:	M18
Submission date:	01/03/2013
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	CSC
Version:	1.4
Status	Final
Author(s):	Mikko Byckling (CSC), Jan Åström (CSC), George Mozdzynski (ECMWF), Mats Hamrud (ECMWF), Jan Westerholm (ABO), Adam Peplinski (KTH), Joerg Hertzer (USTUTT), Konstantinos Ioakimidis (USTUTT), Bastian Koller (USTUTT), Timo Krappel (USTUTT), Rupert Nash (UCL), Timm Krüger (UCL), Adam Carter (UEDIN), Jussi Timonen (JYU), Berk Hess (KTH), Mark Abraham (KTH), Erik Lindahl (KTH)
Reviewer(s)	Alistair Hart (CRAY UK), Erwin Laure (KTH)

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	03/01/2013	Template of the deliverable	Mikko Byckling (CSC), Jan Åström (CSC)
0.2	07/02/2013	IFS added	George Mozdzynski (ECMWF), Mats Hamrud (ECMWF)
0.3	08/02/2013	ELMFIRE added	Jan Westerholm (ABO)
0.4	08/02/2013	Nek5000 added	Adam Peplinski (KTH)
0.5	08/02/2013	OpenFOAM-extend added	Joerg Hertzer (USTUTT), Konstantinos Ioakimidis (USTUTT), Bastian Koller (USTUTT), Timo Krappel (USTUTT)
0.6	08/02/2013	HemeLB added	Rupert Nash (UCL), Timm Krüger (UCL)
0.7	08/02/2013	OpenFOAM added	Adam Carter (UEDIN)
0.75	11/02/2013	Corrections	Mikko Byckling (CSC)
0.8	12/02/2013	HemeLB co-design added	Jussi Timonen (JYU)
0.85	12/02/2013	OpenFOAM corrections	Konstantinos Ioakimidis (USTUTT), Mikko Byckling (CSC)
0.9	14/02/2013	GROMACS added	Berk Hess (KTH), Mark Abraham (KTH), Erik Lindahl (KTH)
1.0	14/02/2013	First version for internal review	Mikko Byckling (CSC), Jan Åström (CSC)
1.1	12/03/2013	Reviewer comments addressed	Mikko Byckling (CSC)
1.2	12/03/2013	Co-design section added to Executive summary	Jan Åström (CSC)
1.3	15/03/2013	Reviewer comments addressed	Mikko Byckling (CSC)
1.4	15/03/2013	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 initial roadmap for the CRESTA codes described in Deliverable D6.1.1. The main progress and main updates to the original roadmaps for the separate codes are summarized in Section 1.1. Actions related to co-design progress for each application are summarized in Section 1.2.

1.1 Summary of progress and roadmap update

The main progress and main updates to the original roadmap for each application can be summarized as follows:

ELMFIRE: The overall progress has been steady, with the majority of the tasks progressing without significant delays. 3D domain composition has been slightly delayed. This may cause a slight delay to the remaining tasks as well, but not by any considerable amount.

GROMACS: The overall progress has been good, with all the tasks progressing as scheduled. A task related to parallel I/O has been rescheduled to finish by M36. Fundamental work on O(N) algorithms for PME electrostatics is ongoing. There are no major updates to the original roadmap.

HemeLB: The overall progress has been good, with all the tasks progressing as scheduled. Several task in pre- and post-processing have been completed. Several successful co-design tasks related to GPU implementation, partitioning and inlet/outlet boundary conditions have been undertaken.

IFS: The overall progress has been excellent, with most of the tasks included in the original roadmap already completed. The code is well ahead of schedule. In order to take advantage of the extra effort available, several additional tasks with strong co-design components have been set up.

NEK5000: The overall progress has been steady, with the majority of the tasks progressing without significant delays. Implementation of the error-estimator has been slightly delayed.

OpenFOAM: The overall progress has been fair, with several tasks being delayed. The test cases have been successfully set up, but attempts to profile the codebase with standard tools have been unsuccessful. The roadmap has been updated to reflect upon these difficulties. Improving pre –and post-processing and mesh refinement steps are currently seen as the main way forward to continue the development towards exascale.

1.2 Summary of co-design progress

The co-design progress for each application can be summarized as follows.

Elmfire/ABO: ABO has participation in the Lattice Boltzmann on GPUs co-design effort with HemeLB and the OpenACC co-design and performance evaluation with Cray.

Gromacs: GROMACS has participation in pre –and post-processing co-design related to I/O with WP5 and code optimization co-design related to new architectures and programming paradigms with WP3.

HemeLB: HemeLB has participation in Lattice Boltzmann on GPUs co-design with ABO, matrix diffusion co-design with JYU and OpenACC co-design with Cray. HemeLB also does co-design related to pre –and postprocessing with WP5.

IFS: IFS has participation in co-array Fortran co-design with CRAY and development environment co-design with Allinea. A task graph co-design work is currently in preparation with KTH.

NEK5000: Nek5000 has participation in global communication co-design with WP3 and OpenACC co-design with Cray.

Openfoam: OpenFOAM has participation in numerical libraries co-design with WP4. A co-design attempt to resolve problems with profiling tools with WP3 and Cray has been recently initiated.

2 Introduction

This document contains roadmaps over the actions needed to develop the CRESTA codes towards exascale performance. The roadmaps of the different codes are presented in the following chapters. The codes 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 that is extensively used for simulation of biomolecular systems. Useful investigation of this kind of systems is typically limited by computational capacity. The limitations concern both system sizes and, in particular, time duration of interesting processes. Also, efficient implementation of ensembles of simulation is needed for gathering 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 that 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 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
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
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 corresponds 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 4^{th} 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 an 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 processor 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. The matrix has been optimized by reducing the second dimension to a constant, which is the number of cells around a given cell to which charges due to gyrokinetic motion can be moved from the given cell. This reduces memory usage significantly but not enough for large-scale simulations. Memory consumption of ELMFIRE will be addressed after the domain decomposition has been fully implemented.

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.

Focus of the work on Elmfire is to continue on basic scalability, mostly related to memory usage. The version initially provided for the project does not implement any spatial domain decomposition that leads to massive memory usage and data duplication. Particles are split between processors but can, during the simulation, be located in any grid cell in the system, leading to massive memory requirements for gathering the charge data and large data transfers when combining the data. Initial proof-of-concept 3D domain decomposition has been already implemented with promising results.

3.1 Summary of initial roadmap

Task	Scheduled date	Status
3D domain decomposition	M18	Ongoing
Load balancing	M24	In planning

Memory usage for binary collisions	M36	In planning
Parallel file writing	M36	In planning

Table 3-1 ELMFIRE: Initial roadmap

Implement a 3D Domain Decomposition: Implementation of a 3D electrostatic grid cell based domain decomposition of the code, so that each processor can only own particles inside its own grid cells.

Load balancing: The 3D domain decomposition will introduce load-balancing issues as the particles are not evenly distributed between all grid cells in the simulation.

Memory usage for binary collisions: Elmfire calculates collisions between randomly chosen particles close to each other in each time step. In order to assess how close particles are to each other, a separate collision grid is set up. By introducing data structures that avoid duplication this will be improved.

Parallelize File Writing: The file writing, which is currently sequential, needs to be parallelized for Elmfire to scale to exascale sized problems.

3.2 Achievements towards ongoing tasks in initial roadmap

Task	Achievement
3D domain decomposition	1D decomposition into 4 domains: memory requirements down by ¼.

Table 3-2 ELMFIRE: Achievements towards ongoing tasks in initial roadmap

3.3 Updated roadmap

Task	Scheduled date	Status
3D domain decomposition	M22	Ongoing
Load balancing	M28	In planning
Memory usage for binary collisions	M36	In planning
Parallel file writing	M36	In planning

Table 3-3 ELMFIRE: Updated roadmap

3.3.1 Implement a 3D Domain Decomposition (M22)

The version originally provided for the project does not implement any spatial decomposition. Particles are distributed evenly among processors but the electrostatic grid data is duplicated in all processors. This prevents scaling to larger grids than approximately 120x150x8 regardless of the number of cores available. For large scale simulations, of e.g. JET or ITER, it would be beneficial to be able to simulate electrostatic grids up to 3000x4000x16 i.e. almost 1500 times larger than today. An estimate for an ITER simulation is that 640 000 cores would be needed for 590 billion particles. With the current version this would require approximately 28TB memory per core.

We are in the process of implementing an electrostatic grid cell based domain decomposition of the code so that each processor can only own particles inside its own grid cells. This will restrict the grid cell data needed in each processor to its own grid cells and a few surrounding grid cells (in order to propagate the particles in time). It should also remove the need to communicate large amounts of data for the charge data, with the downside of having to send particle data between processors in each time step. This task will be completed by M22. The memory requirements are expected to be reduced relative to the number of domains used. The slight delay compared to original roadmap is currently not expected to cause any significant delays to the completion of other tasks.

3.3.2 Improve Load Balancing (M28)

In the current version, load-balancing is not a large problem but it is expected that the 3D domain decomposition will introduce load-balancing issues as the particles are not evenly distributed between all grid cells in the simulation. These need to be investigated and addressed after the initial domain decomposition has been performed. One approach would be to dynamically reallocate the electrostatic grid based on the workload, that is, the size of the grid and the number of particles.

3.3.3 Improve Memory Usage for Binary Collisions (M36)

Elmfire calculates collisions between randomly chosen particles close to each other in each time step. In order to assess how close particles are to each other, a separate collision grid is set up. Currently this uses 10 times the memory it really needs. By introducing data structures that avoid duplication this will be improved.

3.3.4 Parallelize File Writing (M36)

File writing in Elmfire is presently done by all processes sending data to the master process, which then writes the data to disk. For small simulations this is typically not an issue (< 5% of the each time steps goes to writing diagnostics) but it will likely block large scale simulations and input files for visualizations. The file writing needs to be parallelized for Elmfire to scale to ITER sized problems.

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, but 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 scaling 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 20 years of development in the field, and current algorithms are very focused on providing simple interaction forms to reduce the floating-point instruction bottleneck. Historically, runtime for these types of codes was completely dominated by the evaluation of interactions between particles. In principle, this lends itself very well to parallelization, but 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 the code is therefore spending a relatively larger part of time on communication [14].

The work in GROMACS is focused on achieving significant improvements for real applications. Seen from the user side, there are three overall important objectives to advance the state-of-the-art for applications: (i) to reduce the time-step per iteration in order to achieve longer simulations, (ii) to be able to handle much larger application systems to model e.g. mesoscopic phenomena, and (iii) to improve accuracy and results for small application systems through massive sampling.

All three aspects are critically important, but they require slightly different approaches. The wall-clock time for a single time-step iteration is already today in the range of a few milliseconds for some systems, and while we have strategies to improve this further we do not believe this 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, but it will involve challenges related to handling of data when a single master node no longer can control all input and output, both when starting execution and for checkpointing or output. Finally, for small systems, the main approach will be ensemble techniques to handle thousands of simulations that each will use thousands of cores.

4.1 Summary of initial roadmap

Task	Scheduled date	Status
Benchmarking new GROMACS releases, and GPU coding	M9	Completed (v.4.5, v.4.6) Ongoing (development versions)
Multi-grid solvers for efficient PME electrostatics	M36	Ongoing
Efficient large-scale I/O	M27	Ongoing
Task-based parallelism	M36	Ongoing
Ensemble computing & parallel adaptive molecular dynamics	M36	Completed (initial release), Ongoing (development versions)

Table 4-1 GROMACS: Initial roadmap

Benchmarking new GROMACS releases, and GPU coding: GROMACS 4.6 has been released, delivering higher performance and better strong scaling through algorithmic improvements, GPU parallelisation and SIMD kernels.

Multi-grid solvers for efficient PME electrostatics: The implementation of the current standard electrostatics treatment (PME) shows the expected O(N log N) performance scaling with N atoms. Investigating the feasibility of implementing and using known O(N) algorithms is ongoing.

Efficient large-scale I/O: I/O demands of petascale simulations, particularly with algorithms that scale as O(N), will require more efficient I/O techniques. These will come both from using I/O algorithms that do not require explicit communication stages, and better compression formats used before writing output.

Task-based parallelism: Maximising usage of compute resources at exascale will require re-writing algorithms to express their needs and dependencies in a hardware-agnostic way, so that scheduling of tasks can be truly flexible. The associated paradigm shift away from simple procedural flow of GROMACS is a major challenge.

Ensemble computing & parallel adaptive molecular dynamics: To exploit parallelism at the level of multi-simulation ensemble algorithms, we are developing a parallel computing platform Copernicus that uses GROMACS to run massively parallel simulations while doing live intermediate processing to guide new runs.

4.2 Achievements towards ongoing tasks in initial roadmap

Task	Achievement
Benchmarking new GROMACS releases, and GPU coding	GROMACS 4.6 released. Second generation GPU code used in production.
Multi-grid solvers for efficient PME electrostatics	Work in progress.
Efficient large-scale I/O	Collaboration to implement compressed I/O format established.
Task-based parallelism	OpenMP-based threading now used in many parts of GROMACS release 4.6.
Ensemble computing & parallel adaptive molecular dynamics	Copernicus 1.0 has been released, and used for production simulations.

Table 4-2 GROMACS: Achievements towards ongoing tasks in initial roadmap

4.3 Updated roadmap

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

Table 4-3 GROMACS: Updated roadmap

4.3.1 Benchmarking new GROMACS releases, and GPU coding (M18, M30)

GROMACS version 4.6, which has been developed during the first part of the project, has been released. It has brought some important new advances in domain decomposition and scaling over previous versions. We have developed a new set of computational kernels that have departed from the classical implementation with neighbour lists. These have made it much easier to parallelize (both with SIMD and multi-threading), which achieves a higher fraction of the hardware peak floating-point performance. These kernels are also being implemented on GPUs, and GROMACS 4.6 uses heterogeneous acceleration; some kernels run on the GPU while others execute simultaneously on the CPU (where the domain decomposition is also done). Our old style computational kernels have also been upgraded with new SIMD implementations. It will be an important step to benchmark all these new kernels on different hardware, in particular large clusters with GPU co-processors (such as Cray XK6), and in this frame we have also implemented support for the Nvidia Kepler architecture, which is used in several new Cray installations.

4.3.2 Multi-grid solvers for efficient PME electrostatics (M36)

The vast majority of biomolecular simulations rely on particle-mesh Ewald (PME) lattice summation to handle long-range electrostatic interactions. Since this in turn relies on 3D FFTs, the associated all-to-all communication pattern is a major bottleneck for scaling. We are developing improved FFT algorithms and communication patterns, but to improve support for heterogeneous architectures such as CPU-GPU parallelism on each node, we need to develop algorithms that avoid communicating grids over all processors. This can currently be achieved either through multipole [11] -or multigrid-based [12] methods, and we are currently investigating both techniques. This part is targeting "medium" parallelization for normal-size systems (10k-100k cores), and the O(N) algorithms will provide virtually perfect weak scaling, even for systems including long range electrostatics (currently this is only true for simple cut-off interactions).

4.3.3 Efficient large-scale I/O (M36)

With the completion of long-range electrostatics algorithms that exhibit O(N) scaling, it should be possible to reach multi-petascale for normal simulations of very large systems such as virus particles, complexes of several molecules, or material science studies. Typical simulations in this domain might involve a few hundred million particles. To support this, we need to rewrite the input/output layer of GROMACS so that a large set of I/O tasks participate in reading the data from files to avoid running out of memory on the master node, not to mention avoid global communication during startup. This will ideally use a minimalistic PGAS-like library that is fully portable (or even included in the code), so that all I/O code does not have to do explicit communication. We will also implement code for check-pointing and trajectory output that supports asynchronous output by sending the data to a subset of I/O nodes that then transpose the data (to be decomposed over time-frames rather than space), and write it to trajectories while the simulation continues. The completion of this task has been rescheduled from M27 to M36.

4.3.4 Task-based parallelism (M36)

One of the most significant long-term changes will be a complete code re-write to support introduction of task-based parallelism to improve the efficiency inside many-core nodes, to enable better simultaneous utilization of CPU-GPUs, and to enable overlap of computation and communication between nodes. The latter will be particularly critical to increase scaling appreciably, since we are gradually moving into the realm where more time is spent on communication than computation. We have incorporated OpenMP thread-based parallelism into many key performance bottlenecks in GROMACS 4.6, with pleasing results. However, a slightly higher-level abstraction is required to implement task-based parallelism over the whole iterated code. We plan to experiment implementing this with Intel's Threading Building Blocks. Ultimately, such approaches may allow future versions of GROMACS to execute kernels in a hardware-agnostic way.

4.3.5 Ensemble computing & parallel adaptive molecular dynamics (M36)

The true aim of most molecular dynamics simulations is to explore the statics and dynamics of thermal systems, which requires combining results from multiple runs into a single result. This provides an opportunity for parallelism – especially when combined with adaptive sampling methods such as Markov state modelling with parallel adaptive kinetic clustering, or free energy perturbation with adaptive optimizations. In order to fully exploit these adaptive algorithms, we have developed Copernicus, a platform that executes such high-level algorithms while keeping track of many thousands of simulations simultaneously. The parallelism is expressed by combining coarse-grained computable items such as simulations and output processing into a data-flow network, which is executed on workers that may run on a variety of computational resources, including multiple high-performance computing facilities simultaneously. While the coordination is done on Copernicus servers (see Figure 1), the workloads sent to the workers is matched to their capabilities, optimizing total throughput. Each worker unit can be a parallel GROMACS simulation using thousands of cores, and by combining several domains it should be possible to use millions of cores in parallel. 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. We have currently implemented several adaptive sampling algorithms: Markov state modelling, adaptive free energy perturbation, and a string method for minimum free energy pathways in Copernicus. 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 greater than 1M cores feasible.

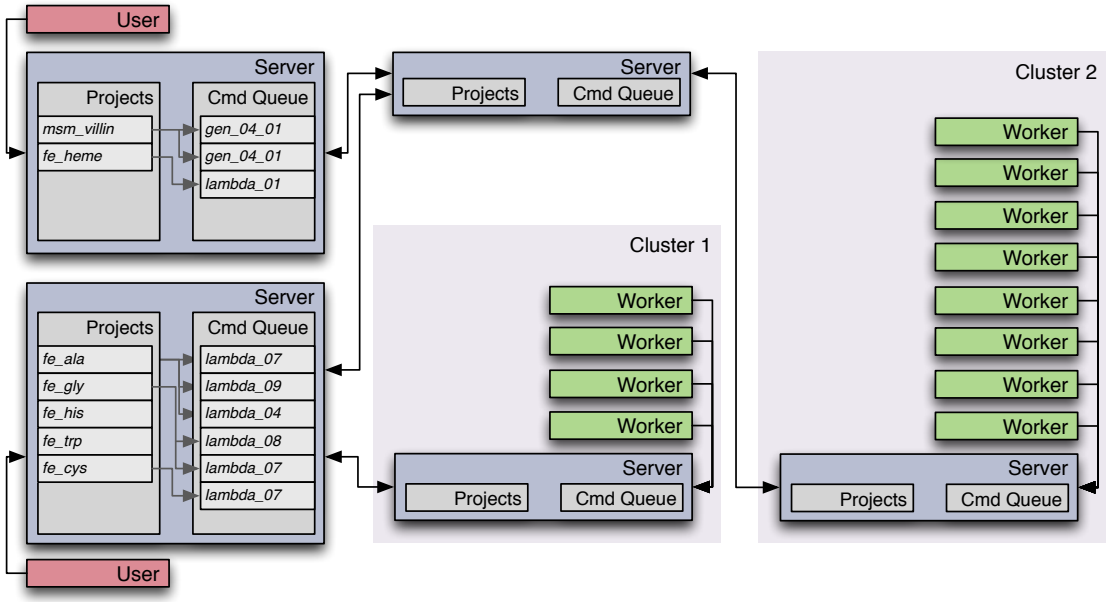

Figure 1 The parallelization task network used in Copernicus.

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 that, due to its locality, is intrinsically easy to parallelise. HemeLB uses MPI for communication and has been shown to have good scalability up to over 32k CPU cores.

5.1 Summary of initial roadmap

Task	Scheduled date	Status
Visualisation and Steering	M36	Ongoing
Pre-processing	M36	Ongoing
Introspection	M36	Ongoing

Table 5-1 HemeLB: Initial roadmap

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.

5.2 Achievements towards ongoing tasks in initial roadmap

Task	Achievement
V&S	Collaborated with WP5 partners to define the visualisation needs of HemeLB.
V&S	We have enabled the coupling of HemeLB to other codes that supply inlet/outlet boundary conditions.
Pre-processing	Enhanced HemeLB setup tool performance by a factor of five with a new algorithm.
Pre-processing	Detailed measurements about the performance of the existing domain decomposition.
Pre-processing	Reduction of initialisation by a complete redesign of the communication patterns for this step.
Pre-processing	Worked with DLR to define the interface to a library (PPStee) that will enable the use of multiple partitioners without significant code changes.
Introspection	Improved performance monitoring functionality of HemeLB by the addition of multiple, efficient timers to the code.

Table 5-2 HemeLB: Achievements towards ongoing tasks in initial roadmap

We have made numerous small enhancements to the scaling of HemeLB.

We have introduced unit testing to the HemeLB setup tool in order to increase confidence and to enable faster development of the software.

As a co-design effort with ABO, a multi-GPU 3DQ19 Lattice Boltzmann code (written in CUDA) which overlaps communication with computation, achieving 180 million lattice point updates per second per GPU has been implemented. When a corresponding code was implemented with OpenACC pragmas to run on multi-GPUs, 160 million lattice point updates per second per GPU was achieved.

As a co-design effort with JYU, two tasks were performed by the JYU group. A tool to extract the simulation geometry from angiographic images was constructed, and an interface was constructed. The interface allows running HemeLB in parallel with some other code that has been parallelized with MPI, so that data can be continuously exchanged between the two codes. In fact the constructed interface is rather generic and facilitates parallel running of two MPI parallelized codes with only minimal changes in these codes.

5.3 Updated roadmap

Task	Scheduled date	Status
Single core performance	M20	On-going
Domain decomposition	M24	On going
Hybrid parallelism	M30	In planning
Steerable parameter extraction	M30	In planning
Visualisation	M36	On going
Introspection	M36	Complete

Table 5-3 HemeLB: Updated roadmap

5.3.1 Single core performance enhancement (M20)

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.

5.3.2 Domain decomposition (M24)

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.

5.3.3 Hybrid parallelism (M30)

Based on the report by Alan Gray (UEDIN) that was the main output of our co-design work [9], 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.

5.3.4 Steerable property extraction (M30)

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 that can be recorded for more detailed off-line analysis.

5.3.5 Visualisation (M36)

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

5.3.6 Introspection (M36)

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.

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 deterministic forecast, a four dimension variational analysis (4D-Var), an ensemble prediction system (EPS) and an ensemble data assimilation system (ENDA).

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 deterministic 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 initial roadmap

Task	Scheduled date	Status
Coarray kernel	M6	Completed
IFS CY37R3 port	M6	Completed
Legendre transform coarray optimization	M15	Completed
IFS CY38R1 port	M15	Cancelled
Semi-Lagrangian coarray optimization	M21	Completed
Optimization of Fourier latitude load-balancing heuristic	M27	Completed
Development of a future solver for IFS	M36	Ongoing
Fourier Transform coarray optimization (added).	M15	Completed

Table 6-1 IFS: Initial roadmap

Coarray kernel: Develop kernel to investigate overlapping computation and communication using Fortran 2008 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.

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

IFS CY38R1 port: Port IFS model (CY38R1) to HECToR. This code cycle became available in 2Q2012 and included support for the T3999 model resolution, Fast Legendre Transform (FLT) and substantially reduced memory requirements for computing the associated Legendre coefficients. Two main subtasks associated with this task are:

1. Run T3999 IFS model (5 km global model)
2. Assess "Legendre transform" optimization at T3999.

This task was cancelled due to delays in the installation of ECMWF's Power7 clusters that were needed for the initial experimentation of the T3999 model resolution at ECMWF. In the course of this early experience (during 4Q2012) some optimizations were made to substantially reduce the cost of model startup and memory use. This work was included in the IFS CY38R2 release that became available during 4Q2012, which will be packaged as the RAPS13 benchmark and ported to HECToR during 1Q2013.

Semi-Lagrangian coarray optimization: Developments to the IFS semi-Lagrangian scheme to use Fortran 2008 coarrays to improve scalability by removing the need to perform full halo wide communications.

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 and was also simulated to beyond 1M cores (assuming 16 threads per task).

Development of a future solver for IFS: Research into a new multigrid solver for extreme scaling of IFS and a potential replacement of the spectral method. Such a solver could be initially tested using a shallow water model code and not IFS. It should be noted that this development is not part of ECMWF's current research plans and should be considered more speculative.

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.

6.2 Achievements towards ongoing tasks in initial roadmap

Task	Achievement
Development of a future solver for IFS	Participation in UK Met Office GUNGHO! project to develop a new dynamical core for their Unified Model. ECMWF has further added two scientists in its Numerical Aspects section to progress this long term development.

Table 6-2 IFS: Achievements towards ongoing tasks in initial roadmap

6.3 Updated roadmap

Task	Scheduled date	Status
IFS CY38R2 port	M18	Ongoing
Investigate GPGPU use in IFS	M27	In planning
Investigate graph based (DAG) parallelization	M36	In planning

Table 6-3 IFS: Updated roadmap

6.3.1 IFS CY38R2 port (M18)

Port IFS model code version CY38R2 to HECToR. This code cycle became available in 4Q2012 and included support for the T3999 model resolution, FLT and substantially reduced memory requirements for computing the associated Legendre coefficients. This code version will be packaged as a RAPS13 benchmark and will include all the IFS Fortran 2008 coarray optimizations implemented in the first year of the CRESTA project. Two main subtasks associated with this task are:

1. Run T3999 IFS model (5 km global model)
2. Assess “Legendre transform” optimization at T3999.

6.3.2 Investigate GPGPU use in IFS (M27)

Some initial experience of using GPGPUs will be made as part of the TITAN INCITE award to the CRESTA project. Specifically we want to explore intercepting the DGEMMs that are called in the Legendre transform and executing them on the GPGPUs. Tests will be performed at the T3999 model resolution to assess the benefit of this approach as applied to the non-FLT and FLT approaches. As the FLT uses a butterfly scheme, each stage of the butterfly has a number of independent DGEMMs that can be run concurrently on the GPGPU with potentially greater efficiency (using NVIDIA’s HYPER-Q mechanism). Finally, as the DGEMMs in the Legendre transform involve matrix multiplication by a per spectral wave number constant matrix, it is hoped that such constant matrices can be located in GPGPU memory, and avoid the need to be loaded every time from node memory.

6.3.3 Investigate graph based (DAG) parallelization (M36)

The use of graph-based parallelization in IFS will be investigated. This work will start by developing a kernel to simulate some of the code features of IFS, including some random delays to simulate load imbalance and operating system interference. A number of DAG approaches will then be tested and assessed for ease of use and performance. Given the code complexity of IFS, it is expected that the graph will be explicitly specified and not created by compiler pre-processing as in some (possibly all) DAG implementations. This work will be a co-design effort with WP3 to draw on existing experience of DAGs in that work package.

7 NEK5000

Nek5000 [1] is an open-source code for the simulation of incompressible flow in complex geometries. The discretization is based on the spectral-element method (SEM) that 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 [2] and on the IBM BG/P Eugene with 262144 cores [3]. An overview of the capabilities and recent developments within the Nek5000 community is given in [4].

7.1 Summary of initial roadmap

Task	Scheduled date	Status
Investigate existing code architecture	M18	Completed
Implement error estimator and initial refinement code	M12	In planning
Adaptive refinement development	M18	Ongoing
Implement load balancing using existing Nek5000 tool suite	M24	In planning
Undertake test and development on large scale applications	M36	In planning

Table 7-1 Nek5000: Initial roadmap

Investigate existing code architecture: To gain a fundamental understanding of most aspects of implementation of NEK5000 with 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. Such error estimators based on the solution of the adjoint equations (dual problem) will be formulated and implemented into 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 are focusing on implementing h-type refinement into NEK5000.

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.

Undertake test and development on large scale applications: By using the developed software environments to conduct simulations of large-scale real-life and industrial application. These applications may include the simulation around a full airplane wing include the transition and separated region, and complex internal flows.

7.2 Achievements towards ongoing tasks in initial roadmap

Task	Achievement

Investigate existing code architecture	Identification of NEK5000 requirements for GPU acceleration and for h-type refinement.
Adaptive refinement development	Integration of NEK5000 pre-processing tools with p4est library completed.

Table 7-2 Nek5000: Achievements towards ongoing tasks in initial roadmap

Investigate existing code architecture: In cooperation with Paul Fischer (the main developer of NEK5000) we have investigated the code architecture and the possible improvements that can be developed within CRESTA. Two main directions of development have been chosen: porting NEK5000 to GP-GPU and introducing adaptive mesh refinement (AMR). For GP-GPU port the simplified version of the code NEKBone has been selected. On the other hand we have decided to use the p4est library [15] for the management of AMR tree grid. A test suite has been done as a part of this task.

Adaptive refinement development: There are two ways of introducing AMR: adaptive p-refinement, i.e. increasing polynomial order in element, and adaptive h-refinement, i.e. splitting the element into smaller one. After discussion with Paul Fisher we have discarded p-refinement in favor of h-refinement, due to its flexibility. However, in this case additional software managing variable grid structure is necessary. For this we are going to use p4est library. Currently we have completed integration of NEK5000 pre-processing tools with p4est library.

7.3 Updated roadmap

Task	Scheduled date	Status
Adaptive refinement development	M18	Ongoing
Implement error estimator and initial refinement code	M24	In planning
Implement load balancing using existing Nek5000 tool suite	M30	In planning
Undertake test and development on large scale applications	M36	In planning
OpenACC acceleration of Nek5000	M27	Ongoing

Table 7-3 Nek5000: Updated roadmap

7.3.1 Adaptive refinement development (M18)

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 are focusing on implementing h-type refinement into NEK5000.

7.3.2 Implement error estimator and initial refinement code (M24)

Adaptive mesh refinement requires identification of the regions in the flow with significant error. Such error estimators will be formulated based on the solution of the adjoint equations (dual problem) that can be thought as a measure of the sensitivity of certain observables to the local mesh quality. We are going to create an interface between the error estimator and NEK5000. Due to the change in the developed AMR type (h- versus p-refinement) this task has been postponed.

7.3.3 Implement load balancing using existing Nek5000 tool suite (M30)

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 are going to investigate and analyze the load balancing affected the refinement using the existing analysis tools developed by WP3 Task 3.3.

7.3.4 Undertake test and development on large scale applications (M36)

Using the developed software environments we are going to conduct simulations of large-scale real-life and industrial application. These applications may include the simulation around a full airplane wing include the transition and separated region, and complex internal flows. Such simulations are only possible with adaptive mesh refinement, proper boundary treatment and adapted post-processing tools, all aspects to be developed during CRESTA

7.3.5 OpenACC acceleration of Nek5000 (M27)

The objective of the task is to enable, for the first time, the use of Nek5000 on massively parallel hybrid GPU/CPU system. This task will firstly show how to efficiently use hybrid massively parallel computing on a simplified version of Nek5000. And then it will outline the possible paths to speed up the full Nek5000 with GPU. In addition, this project will assess the effectiveness of OpenACC and compiler support for GPU programming with a view to future hybrid exascale simulations.

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, different parties have made modifications to the code 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.

It is hoped that any changes to the code contributed by the CRESTA project could be made available for inclusion in *both* distributions, but if there are good reasons to make optimizations or improvements to one particular version, we will do so.

Since the code can be used in many different ways, it is challenging to identify ways to enable the application for exascale systems in general. It is likely that there are some problems that are much more amenable to large-scale systems, but it is not obvious *a priori* that there is much to be gained in making simulations of “simple” systems (such as Lid-driven Cavity Flow) scale to many more processors than at present.

It has been observed that with OpenFOAM, in many cases the bottleneck does not lie in the actual solution phase, but in the pre –and post-processing stages of the computation. It is currently being investigated how to efficiently parallelize these phases of the solution process in order to enable the scaling of OpenFOAM to exascale geometries and meshes.

8.1 Summary of initial roadmap

Task	Scheduled date	Status
Benchmarking of the latest version of the code	M12	Ongoing
Code analysis of latest version of code	M12	On hold
Performance analysis of kernels, libraries	M15	Delayed
Iterative performance improvement	M24	Delayed
Test case 01: ercoftac Square Cylinder with OpenFOAM-1.6.ext	M10	Completed
Test case 02: pump turbine power plant with OpenFOAM-1.6.ext	M11	Completed
Test case 01: ercoftac Square Cylinder with OpenFOAM-2.1	M13	Completed
Test case 02: pump turbine power plant with OpenFOAM-2.1	M18	Ongoing

Table 8-1 OpenFOAM: Initial roadmap

Benchmarking of the latest version of the code: Detailed benchmarking of the code has proved considerably more difficult than what was initially assumed. Scaling results for a number of different test cases have been obtained, but these measurements relate only to the total run-time of the code. Initially, even these runs proved to be non-

trivial, mostly for reasons related to the multiple steps involved in running a test case (some of which have to run in serial, and others in parallel) and also due to the fact that not all test cases were fully documented. Efforts to obtain more detailed profiles for the code have proved unsuccessful to date. Standard tools such as CrayPAT have been unable to cope with the highly-templated C++ codebase, often crashing during the code instrumentation stage. For this reason, it has not yet been possible to measure the performance of the code in detail.

Code analysis of latest version of code: This task was started, but it quickly became evident that the code was too large and complex to understand in the available time without such investigations being supported and driven by results from the benchmarking. Since these results have not yet been obtained, it has not been possible to extract from the code those sections that are of interest from the point of view of performance and scalability to exascale problem sizes.

Performance analysis of kernels, libraries: For the same reasons as described above, a detailed analysis has not been possible. This task will be readdressed once a better overall understanding of the performance of the code as a whole has been obtained.

Iterative performance improvement: This task has also been delayed until more detailed performance analysis can be undertaken.

Test case 01: (ERCOFTAC Square Cylinder with both OpenFOAM-1.6-ext and OpenFOAM-2.1) To check if the physics is correct and independent from the large number of cores used for the simulation run, we have prepared the ERCOFTAC square cylinder with about 15 million grid vertices. The ERCOFTAC square cylinder is a unique test case that is experimentally measured [7]

Test case 02: (Pump turbine power plant with both OpenFOAM-1.6-ext and OpenFOAM-2.1) The pump turbine test case was prepared for the OpenFOAM-1.6-ext version. It has been never been released for performance tests due to OpenFOAM-1.6-ext not being compiled on HERMIT. For future work OpenFOAM-2.1 will be used, and therefore the test cases are being converted to the new format to be ready for performance and scalability tests.

8.2 Achievements towards ongoing tasks in initial roadmap

Task	Achievement
Benchmarking of the latest version of the code	Scaling results for a selection of cases including damBreakFine, damBreak3D, motorBike and motorBikeLarge covering the simpleFoam and interFoam binaries.
Test case 01: ercoftac Square Cylinder with OpenFOAM-1.6.ext	Adequate physical results. Important for comparison with OpenFOAM-2.1 in the future.
Test case 02: pump turbine power plant with OpenFOAM-1.6.ext	Adequate physical results. Important for comparison with OpenFOAM-2.1 in the future. See Timo Krappel [8]
Test case 01: ercoftac Square Cylinder with OpenFOAM-2.1	Adequate physical results. In agreement with OpenFOAM-1.6.ext

Table 8-2 OpenFOAM: Achievements towards ongoing tasks in initial roadmap

To check if the physics is correct and independent from the number of cores used for the simulation run, we have prepared the ERCOFTAC square cylinder with about 15 million grid vertices. The ERCOFTAC square cylinder is a unique test case that is experimentally measured [7]

The flow concerned in the first test case corresponds to turbulent flow of water around a square cylinder. The size of the square cylinder (H) is 0.04m and it extends along the width of the channel, the cross-section of which is 0.40x0.56m. The mean velocity at the inlet, U, is assumed to be 0.535m/s and it is taken as reference value. The Reynolds number, based on U and H, is 21400. The shedding frequency, f, is estimated experimentally to be 1.77Hz. The resulting Strouhal number (St=f H/U) is 0.132. The flow produced in that manner is very interesting to use for a Large Eddy Simulation (LES), since it involves coherent shedding of vortices from the cylinder. Further details about the flow and this test case can be found on the ERCOFTAC web site (see test case 43) [7]. This is the first test case that IHS has prepared for performance and scalability tests of the OpenFOAM® solver, pimpleFOAM.

pimpleFOAM stands for piso-simple-FOAM and is a combination of the piso and simple algorithms, which are used for the pressure-velocity coupling. Computations made by IHS (Timo Krappel) show that the pimple algorithm is more stable than the piso algorithm.

8.3 Updated roadmap

Task	Scheduled date	Status
Benchmarking of the latest version of the code	M18	Ongoing
Code analysis of latest version of code	M21	Planned
Performance analysis of kernels and libraries	M24	Planned
Iterative performance improvement	M27	Planned
Test case 02: pump turbine power plant with OpenFOAM-2.1	M18	Ongoing
Test case 01: mesh refinement	M20	Ongoing
Test case 02: mesh refinement	M22	In planning

Table 8-3 OpenFOAM: Updated roadmap

8.3.1 Benchmarking of the latest version of the code (M18)

Version 2.1.0 of OpenFOAM has been released since the CRESTA project started. There have been some fairly major changes to the code since version 1, including the incorporation of parallel mesh generation. Benchmarking and profiling of OpenFOAM have been undertaken on previous versions, but before we know where to concentrate our efforts in optimization for future systems, we need to understand the impacts of recent changes on the code's performance.

In addition to providing an update of previous results on the performance of OpenFOAM based on current systems and the newest version of the code, we will adjust parameters of our profiling runs in order to attempt to measure how the performance would vary as the ratios of computation, communication and memory access vary. In addition, we will specifically investigate the I/O performance of the code and seek to identify how these I/O patterns are likely to change when scaling up to exascale.

8.3.2 Code analysis of the latest version of the code (M21)

In tandem to measuring the performance of the code, an analysis of the code's structure will be undertaken in order to, for example:

(i) determine internal interfaces in the code where alternative solvers, libraries, etc. could be swapped in if it was determined that these could provide better performance;

(ii) determine the parallelisation patterns currently used in the code and evaluate these with respect to exascale issues such as fault-tolerance. A simple example of this might be that a synchronous domain-decomposition might not be intolerant to a process failing, whereas a tracked task-farm approach might be able to recover from a process failing. (Note that this is example is illustrative. At present, there is no evidence that either of these patterns is directly relevant to OpenFOAM.)

8.3.3 Performance analysis of kernels, libraries (M24)

In the course of the activities above, we will have been able to quantitatively measure the characteristics of the sub-problems solved by libraries and routines used for linear algebra and meshing. We will then engage with the developers of these libraries and seek comparisons with the other applications investigated in WP6 to determine possible optimisations.

8.3.4 Iterative performance improvement (M27)

Concentrating on those parts of the code which have been determined to be potential future bottlenecks, we will use standard optimisation techniques to seek to improve the scaling of the code (including, for example, overlapping communication and computation, possibly through the use of more asynchronous communications, investigating the effects of compiler optimization, changing memory access patterns, introducing further (hybrid) levels of parallelisation). For pre –and post-processing, current sequential tools may have to be replaced with new parallel ones to enable working with exascale meshes and geometries.

8.3.5 Test case 02: pump turbine power plant with OpenFOAM-2.1 (M18)

The geometry of the test case 02 is shown below in Figure 2.

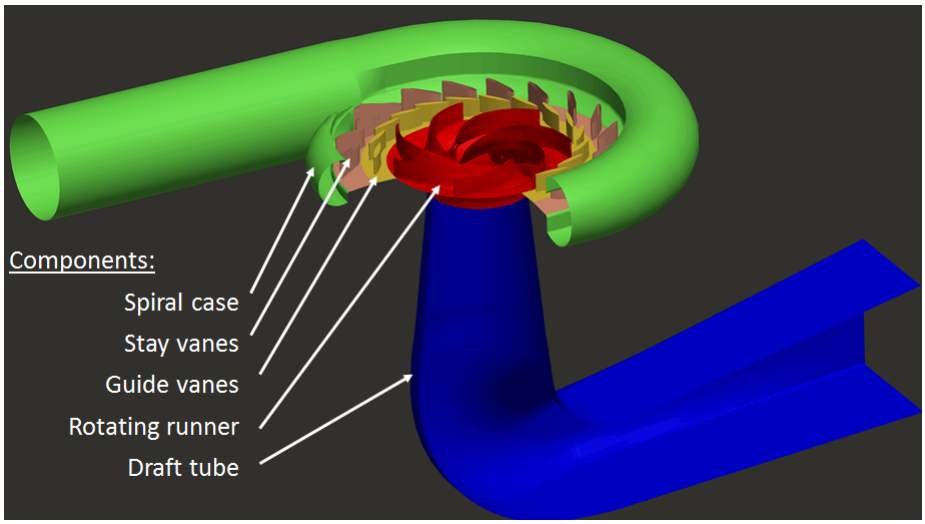

Figure 2: Geometry of a Francis turbine

The application of OpenFOAM® at the Institute of Fluid Mechanics and Hydraulic Machinery, University of Stuttgart, is the simulation of the flow in an entire hydraulic turbine using a Large Eddy Simulation (LES). This means that a great part of the turbulence in the flow will be resolved in the computation up to very fine turbulent scales. Since the Reynolds number of this flow is very high this simulation needs very fine computational grids, very fine time steps and long simulation times. Consequently a very high computational effort is required. According to a publication of Chapman [5] and Fröhlich [6] the number of vertices in the computational domain can be estimated to approximately 1000 million for all parts of a hydraulic machine.

The test case consists of stationary and moving parts, both with a complex geometry. The runner rotates so we have the non-trivial case of rotor-stator interaction.

Furthermore, in every part of the hydraulic machine we have turbulence phenomena that cover the whole range and scales of turbulence. Another topic very difficult to resolve is the vortex rope which is produced by the runner as it rotates in the draft tube.

The extension of the solver pimpleFOAM to dynamic mesh also exists, it is called pimpleDyMFOAM. This is the solver to be used in the future work in order to compute the whole pumpturbine. Another reason to use the pimple algorithm is that the piso algorithm does not give accurate results for this machine.

A new general Grid Interface (GGI) called Arbitrary Mesh Interface (AMI) exists in OpenFOAM-2.1. It seems to offer a better performance than the GGI implementation of OpenFOAM-1.6-ext. Furthermore, the use of OpenFOAM-2.1 allows compatibility with other OpenFOAM® users and developers in the CRESTA framework.

Furthermore, the standard simulation technique in OpenFOAM® for incompressible flows is an implicit time discretization with a SIMPLE or PISO type pressure-velocity coupling. These algorithms could be computationally time expensive because of the need to repeatedly solve global systems of linear equations in an iterative loop. The solution of these global linear equation systems could be a bottleneck for a LES on very fine grids. Performance and scale up tests will be carried out in order to identify if the algorithms mentioned before are able to get good results, as well as a good performance with OpenFOAM-2.1. If it is not the case, the algorithms will be changed towards an explicit formulation. A version of the Fractional Step Method would be proposed to solve the equations. It is well known that the Fractional Step Method (FSM) is used for Direct Numerical Simulation (DNS) and LES to enhance the stability of the solution. It is expected, that this method will reach a higher performance for very large computational grids.

8.3.6 Test case 01: mesh refinement (M20)

In OpenFOAM® uses various utilities for mesh preprocessing. For our usage scenarios, the utilities required are *decomposePar, refineMesh* and *mapFields*.

The *decomposePar* utility is used for the domain decomposition and is coupled with the PT-Scotch [10] library. To our knowledge, partitioning of extremely large meshes with the PT-Scotch library has been performed for academic cases, our purpose is to investigate the usability of the library for real-world problems. Furthermore, either the refinement tool *refineMesh*, or the *mapFields* utility, are not parallelized. It is essential to perform analysis with these tools and check for possible bottlenecks.

The mesh refinement effort enables co-design opportunities with partners in WP3, WP4 and WP5. It will be paramount to parallelize *refineMesh* and *mapFields* tools in order to perform simulations at an exascale.

8.3.7 Test case 02: mesh refinement (M22)

The relevant test case (Testcase 02) is the Large Eddy Simulation (LES) of the flow in an entire hydraulic machine. A realistic simulation requires very fine computational grids and consequently a very high computational effort.

The flow in a hydraulic machine is characterized by relative high Reynolds numbers (Highly Turbulent Flow) in test rig of about $3*10^6$ to $5*10^6$. We estimate that the number of vertices (from 750000 to a billion vertices) and the small time step size will lead to a requirement of approximately 80 million core hours to get a fully converged simulation. This would mean a usage of 60000 cores for about 60 days.

9 References

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

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

[3] 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.

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

[5] Chapman D., "Computational Aerodynamics Development and Outlook", AIAA Journal, Vol. 17, No.12, pp. 1293-1313, 1979.

[6] Fröhlich J., "Large Eddy Simulation turbulenter Strömungen", Teubner Verlag, 1. Auflage, 2006.

[7] ERCOFTAC web site. Available online at: http://www.ercoftac.org/

[8] 7th OpenFOAM Workshop web site. Available online at: http://www.openfoamworkshop.org/2012/OFW7.html/

[9] Alan Gray, "A Feasibility Study on the Hybridisation of HemeLB", CRESTA internal report, 2013.

[10] François Pellegrini, "Scotch & PT-Scotch", 2013. Available online at http://www.labri.fr/perso/pelegrin/scotch/

[11] Greengard, L., Rokhlin, V., "A fast algorithm for particle simulations", J. Comput. Phys. 73, 325, 1987.

[12] Izaguirre, J.A., Hampton, S.S., Matthey, T., "Parallel multigrid summation for the N-body problem", J. Parallel Dist Comp 65, 949-962, 2005.

[13] 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.

[14] 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), pp 435–447, 2008.

[15] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas, "p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees", SIAM Journal on Scientific Computing, 33(3), pp 1103-1133, 2011.