# D6.1.1 – Roadmap to exascale (Initial release)

## *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:** | M6 |
| **Submission date:** | 31/03/2012 |
| **Project start date:** | 01/04/2012 |
| **Project duration:** | 36 months |
| **Deliverable lead organization** | CSC |
| **Version:** | 1.0 |
| **Status** | Final |
| **Author(s):** | J.A. Åström (CSC), Adam Carter (EPCC), Konstantinos Ioakimidis (USTUTT), Rupert W. Nash (UCL), James Hetherington (UCL), George Mozdzynski (ECMWF), Artur Signell (ABO), Jan Westerholm (ABO) |
| **Reviewer(s)** | Stefano Markidis (KTH), Stephen Booth (EPCC) |

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

## Version History

| Version | Date | Comments, Changes, Status | Authors, contributors, reviewers |
|---|---|---|---|
| 0.1 | 02/01/2012 | First version of the deliverable | J.A. Åström (CSC) |
| | | Gromacs added | |
| 0.2 | 23/02/2012 | OpenFOAM added | Joerg Hertzer (USTUTT) |
| | | IFS added | |
| | | ELMFIRE added | |
| | | HemeLB added | |
| | | NEK5000 added | |
| 1.0 | 27/03/2012 | Final corrections made to document based on reviewer feedback | J.A. Åström (CSC) |

# Table of Contents

# 1 Executive Summary

The ‘Roadmap to Exascale’ contains an overview of how the CRESTA applications codes may be developed such that they can take advantage of future computers with computational capacity in the exaflop/second realm. The summary for the codes are as follows:

**ELMFIRE:** needs to be developed such that it includes a true domain decomposition scheme. With the present algorithm it does not seem likely that the code can run efficiently on exascale systems, as it would require unrealistic amount of memory. Also the data handling needs to be made parallel in order to cope with exascale data.

**GROMACS:** is going to be developed towards exascale according to three different strategies: (i) improve wall-clock-time/iteration (ii) soft-scaling improvements for large simulations and (iii) ‘ensemble’ approach for efficient simulations of systems with large fluctuations for which large amount of statistics and/or optimization is needed. These approaches include efficient implementation of GP-GPU algorithms, O(N) FFT and parallel I/O.

**OpenFOAM:** At EPCC the development will focus on the most recent version of OpenFOAM from the OpenFOAM foundation [1].

At the Institute of Fluid Mechanics and Hydraulic Machinery, the University of Stuttgart version OpenFOAM-extend-1.6 [2] will be used. The code has been extended by additional features, including the General Grid Interface (GGI). By these extensions OpenFOAM has become a powerful open source CFD software package specialized in turbo machinery. The interest here will be to simulate a whole hydraulic machine on exascale architectures with the OpenFOAM version mentioned above.

**NEK5000:** will be developed towards exascale scalability by implementing new theoretical solutions for parallelism like: adaptive refinements, alternative discretisation and hybrid parallelisation. Extra care will be taken with respect to exascale boundary conditions, data handling and load balancing.

**IFS:** will be developed by utilizing Fortran co-arrays to overlap calculations and communication for Legendre transforms and semi-Lagrangian halo calculations. Load balancing for Fourier transforms is another area that will be optimized.

**HemeLB:** is going to need an improved visualization scheme for handling exascale data sets. Also the meshing procedure will be improved.

# 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 chapter 3. 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 are needed for gathering statistic validity.

**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. Within this project the focus of University Stuttgart will be on a specialized code for turbo machinery. The objective is to simulate a whole hydraulic machine on exascale architectures.

**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.

**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.

**HemeLB:** is being developed and is intended to form part of a clinically deployed exascale virtual physiological human. HemeLB simulate blood flow in measured blood vessel geometries. The objective is to develop a clinically useful exascale tool.

## 2.1 Glossary of Acronyms

| | |
|---|---|
| **JET** | A Tokamak fusion reactor |
| **ITER** | A Tokamak fusion reactor |
| **CFD** | Computational fluid dynamics |
| **ECMWF** | European Centre for Medium Range Weather Forecasts |
| **ns** | Nanosecond |
| **CPU** | Central processing unit |
| **PETSC** | A computer code algorithm library |
| **GPU** | Graphics processing unit |
| **SIMD** | Single instruction multiple data |
| **PME** | Particle-Mesh Ewalds |
| **FFT** | Fast Fourier transform |
| **O(N)** | Order - N |
| **LES** | Large-Eddie simulation |
| **GGI** | General grid interface |
| **SIMPLE** | Pressure-velocity coupling in CFD |
| **PICO** | Pressure-velocity coupling in CFD |
| **FSM** | Fractional step method |
| **SEM** | Spectral element method |
| **FLT** | Fast Legendre transform |
| **API** | Application programming interface |
| **DSL** | Domain specific language |
| **SL** | Semi-Lagrangian |

# 3 Roadmaps for the codes

## 3.1 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 real time.

Today 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

The most CPU heavy part of the code presently 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 (#grid cells x #grid cells) for collecting charge data for the grid cells. 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. It also introduces an extra index conversion when gathering the data.

Once the grid cell charge data has been combined and split among the processors, each processor can construct its own part of the Poission 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 initial work on ELMFIRE will be on basic scalability, mostly related to memory usage. The version 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. This currently completely prevents simulations on large grids.

The items mentioned below are what currently have been identified as problems preventing ELMFIRE from scaling to simulations larger than 100 000 processors. It is however expected that we find additional, and more important, problems once the initial domain decomposition has been done.

#### 3.1.1 Implement a 3D domain decomposition

The version 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 plan to implement an electrostatic grid cell based domain decomposition of the code so that each processor can only own particles inside its own grid cells. This should 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 amount of data for the charge data with the downside of having to send particle data between processors in each time step. We expect to have implemented the domain decomposition by M18.

#### 3.1.2 Improve load balancing

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. We expect to have measured and implemented load balancing by M24.

#### 3.1.3 Improve 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. Currently this uses 10 times the memory it really needs. By introducing data structures that avoids duplications this could be improved. We plan to implement this by M36

#### 3.1.4 Parallelize file writing

File writing in ELMFIRE is presently done by all processes sending data to process 0, 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. We expect to have implemented parallel file writing by M36.

## 3.2 GROMACS

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 wallclock 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 a parallelization algorithm point-of-view, 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.

### 3.2.1 Benchmarking new GROMACS releases, and GPU coding (Q1-3 2012)

GROMACS version 4.6, which has been developed during the first part of the project, is currently in the beta stage, and will bring 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 neighbor lists, which will make it much easier to parallelize both with SIMD and multithreading, and achieve a higher fraction of the hardware peak floating-point performance. These kernels are also being implemented on GPUs, and Gromacs 4.6 will use heterogeneous acceleration with some kernels running on the GPU while other execute simultaneously on the CPU, where the domain decomposition is also done.

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 will also implement support for the next-generation Nvidia Kepler architecture scheduled for release in spring – these cards in particular will be used on several new Cray installations.

### 3.2.2 Multi-grid solvers for efficient PME electrostatics (Q2 2012-2014)

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 in 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-based [20], or multigrid-based [21] methods, and we intend to investigate both. 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).

### 3.2.3 Efficient large-scale IO (2013)

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 IO 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 start-up. This will ideally use a minimalistic PGAS-like library that is fully portable (or even included in the code), so that all IO 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 IO nodes that then transpose the date (to be decomposed over time-frames rather than space), and write it to trajectories while the simulation continues.

### 3.2.4 Task-based parallelism (2013-2014)

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. At this point we will also investigate the usage of lower-level communication libraries to improve scaling further. Presently, our preliminary tests indicate that automated tools such as OpenMP do not provide sufficiently fine-grained control over the execution, and we might therefore have to use threads directly unless partners in the project come up with better alternatives.

### 3.2.5 Ensemble computing & parallel adaptive molecular dynamics (2012-2014)

Our main disruptive long-term path to true exascale performance will be to combine direct domain-decomposition scaling in individual simulations with ensemble approaches to support simultaneous execution of thousands of coupled simulations. This will be accomplished by using Markov State Models and kinetic clustering for parallel adaptive simulation [Pande, V.S., Beauchamp, K, Bowman G.R., Everything you wanted to know about Markov State Models but were afraid to ask., Methods 52, 99-105 (2010)]. In contrast to the distributed computing approach used e.g. in Folding@Home, exascale resources will enable extremely tight coupling between simulations each using 1k-100k cores. This will make it possible to employ kinetic clustering for slow dynamics (e.g. multi-millisecond structural transitions in proteins) where even single state transitions will require petascale-level simulations, and complete mapping of the processes is simply not possible with todays resources. This will initially be implemented as a separate layer of code, where our idea is to formulate dynamic data flow networks that execute a set of simulations, perform analysis, and based on the result of the analysis a second generation of simulations is executed. The advantage of this approach is that the resulting code will be very easy to adapt to other simulation programs (in principle anything that relies on sampling). In particular, this setup will enable is to achieve exascale performance for *typical application systems.* A target setup is a normal membrane protein system with around 250,000 atoms. With the new electrostatics solvers and task parallelism, we expect to achieve efficient scaling over 1k-10k cores (including heterogeneous CPU-GPU parallelism), and an ensemble could then typically include 1,000 such simulations, which means efficient use of well over a million cores. Larger systems will enable us to push this to even larger supercomputers, and approach a billion cores on future exascale resources. Figure 1 illustrates the multi-level parallelization.

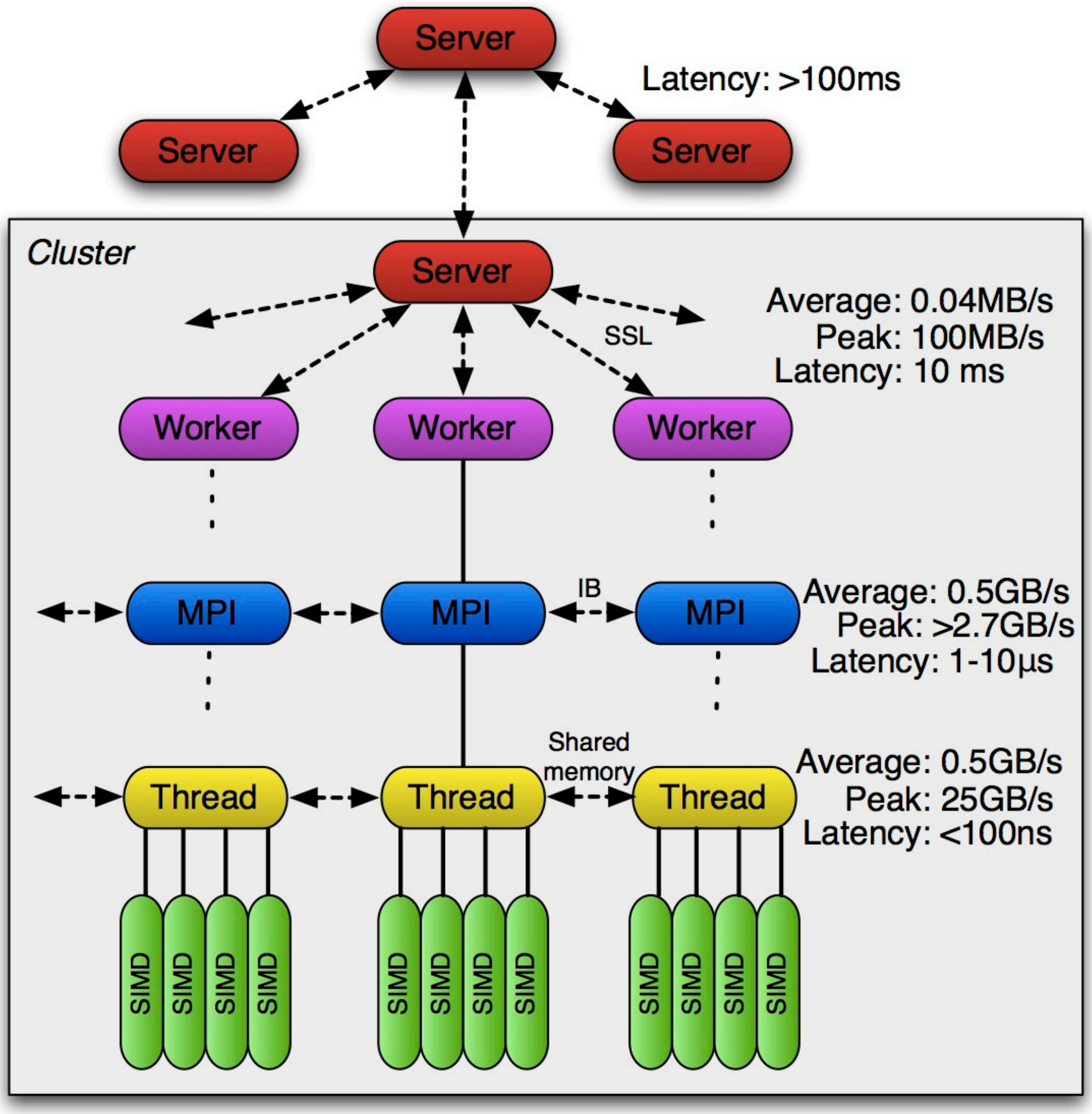

Figure 1: Multi-level parallelism in ensemble-centric parallel adaptive molecular dynamics. "Worker" simulations in a single cluster are members of a tightly coupled ensemble, but it is also possible to couple multiple separate clusters.

## 3.3 OpenFOAM

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. OpenFOAM is licenced 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 organisation, 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 optimisations or improvements to one particular version, we will do so. For example, there is code specific to the Extend project for dealing with moving geometries. If it turns out that this code introduces a performance bottleneck, then this would be a valid candidate for optimisation during the project.

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. In conjunction with contacts at OpenCFD Ltd., we have identified a use-case that is considered a realistic candidate for simulation at exascale. This specific example, which consists of modelling the flow of air around a motorbike, is representative of a wider class of problems that could benefit from simulation on exascale systems.

There is no published roadmap for the development of OpenFOAM, so this activity will have to be fairly reactive to any developments in the releases of the code. Having said that, it is expected that the following approach will be taken to prepare OpenFOAM for exascale systems:

### 3.3.1 Benchmarking of the latest version of the code (2-3Q2012)

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.

### 3.3.2 Code analysis of the latest version of the code (2-4Q2012)

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

- 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;
- 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.)

### 3.3.3 Performance analysis of kernels, libraries (3-4Q2012)

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.

### 3.3.4 Iterative performance improvement (2013)

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).

### 3.3.5 Investigation of alternative parallelisation approaches (2014)

This is a riskier approach to improving parallel performance scaling, but potentially has large rewards, especially if it emerges that future architectures look like they will be qualitatively different from those of today. With a large code like OpenFOAM, it is very difficult to make non-incremental changes to the code, but having gained a good understanding of the code's structure and performance over the first two years of the project, it is likely that proof-of-concept code could be written to demonstrate alternative parallelisation patterns that could eventually be adopted by the code's developers. These will probably involve exposing more potential parallelism in the problem so that the code can make use of the millions of cores expected to feature in the machines of the future. Such patterns could include hybrid message-passing / shared memory approaches, adding task parallelism, or re-computing certain data to reduce communications.

#### 3.3.6 Hydraulic machinery

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 [3] and Fröhlich [4] the number of vertices in the computational domain can be estimated to approximately 1000 million for all parts of a hydraulic machine.

In order to do LES for a whole hydraulic machine (including rotor/stator interaction) the General Grid Interface (GGI) implemented in OpenFOAM is needed. For this reason the version OpenFOAM-extend-1.6 [2]is required. In our knowledge no work has been done on exascale systems with the OpenFOAM-extend-1.6 version. GGI was a bottleneck in the OpenFOAM-extend version but due to a new implementation performs well when running on 512 cores. Further performance and scale up tests will be carried out to find out if GGI is a possible bottleneck on exascale systems. In case GGI could be bottleneck on exascale systems, an upgrade must be carried out.

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-extend. 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.

To realize the tests mentioned above two test cases have been prepared at IHS. To check if the physics is correct quite quickly, we have prepared the ERCOFTAG square cylinder with about 15 million grid vertices. The ERCOFTAG square cylinder is a unique test case that is experimentally measured [5].

Furthermore, the final scope is to compute a whole hydraulic machine and therefore we have as final test case a whole hydraulic machine shown below:

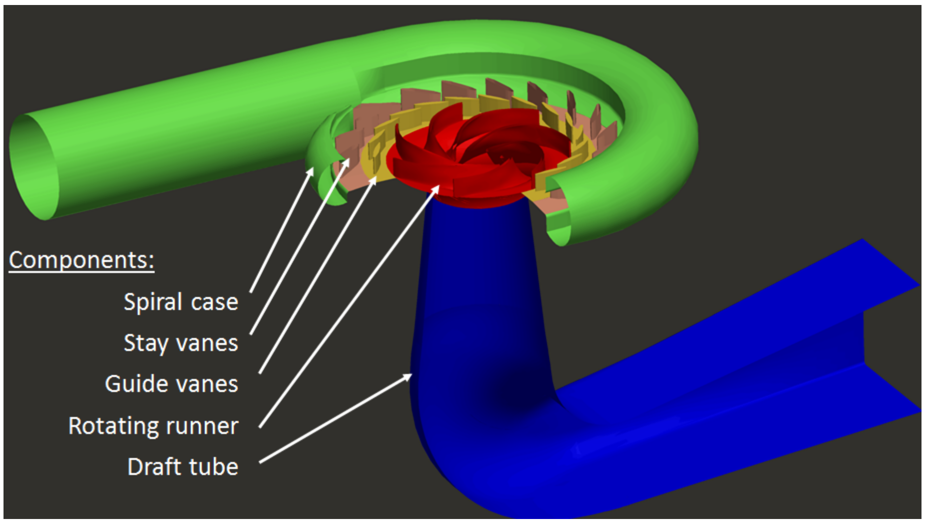

Fig.1: Geometry of a Francis turbine

## 3.4 NEK5000

Nek5000 (nek5000.mcs.anl.gov) 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 *("Coherent structures and dominant frequencies in a turbulent three-dimensional diffuser"* by J. Malm, P. Schlatter and D. S. Henningson, J. Fluid Mech. 2012) and on the IBM BG/P Eugene with 262144 cores ("*Extreme Scaling Workshop 2010 Report*"). An overview of the capabilities and recent developments within the Nek5000 community is given in the presentation by Paul Fischer, Main developer (http://www.mcs.anl.gov/~fischer/nek5000/fischer_nek5000_dec2010.pdf)

Based on the description of Nek5000 in the WP6 needs analysis deliverable D 6.2 in the CRESTA project, we main focus on the development of the following software environment and tools.

### 3.4.1 Adaptive refinement

The original Nek5000 code uses uniform order of the spatial interpolations throughout the domain. The principal way for grid refinement is by global p-refinement, i.e. by increasing the approximation order globally. Adaptive h-refinement, i.e. the splitting of cells into smaller ones, is not possible due to algorithmic considerations (negative effect on scalability). However, local refinement, either adaptive or by user intervention, is a desirable feature for nek5000 which will be crucial for the future scalability of the code, in particular for the simulation of large-scale problems involving turbulence.

In the CRESTA project we will work on framework of adaptive refinement in p-types (various order accurate). The basic idea is that the refinements are only used in the regions with significant errors. Such error estimators can 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. Such estimators have been developed at KTH. Though consideration of multiple local observables such as drag, shedding frequency etc. it is proposed to decide when to switch from lower-order to higher-order (or vice versa).

### 3.4.2 Alternative discretisation

So far, nek5000 is designed to have a spectral-element discretization in all directions, either 2D or 3D. For certain cases, in particular flows in which spatial homogeneity can be assumed in at least one direction, the SEM discretization could be replaced by a more optimal Fourier-Galerkin discretization. A substantial gain in performance can be expected for such flow cases. The algorithmic changes implied by this new discretization, and in particular the impact on scalability will be studied within CRESTA.

### 3.4.3 Hybrid parallelization

In the present state, Nek5000 does not employ any hybrid approach to parallelization. All communication is handled by MPI, which has proven to be very efficient, mainly due to the element structure of the mesh. However, in the light of alternative discretization that might include an additional level into the mesh topology, a hybrid approach should be reconsidered. This will be done in collaboration with WP6 and WP3.

### 3.4.4 Boundary conditions for exascale computing

The definition of boundary conditions requires special attention, especially in cases where large parts of the domain are in the turbulent state. In particular for exascale computations, which are aimed at realistic geometries in large domains, a faithful prescription of boundary conditions is crucial. The challenge is two-fold: First, reflections in the form of pressure waves need to be avoided at boundaries, and secondly, proper convective properties need to be maintained as to reduce the upstream influence of the condition, even in the presence of highly unsteady flow towards the boundary. Similar issues need to be dealt with at inflow boundaries when transient turbulent velocity profiles are required: Simply adding random fluctuations to the DNS profiles lacks the temporal and spatial correlation of real turbulence. The fluctuations must be pre-computed and stored in a database or computed on the fly from an auxiliary computation. In the framework of exascale simulations, the handling of such unsteady conditions needs to be assessed and refined.

### 3.4.5 Pre- and post-processing

In the CRESTA project it is decided to focus on p- instead of h-type refinements. This means that we do not need to consider the mesh generations. However, only quadrilateral (2D) and hexahedral (3D) elements are used in the types of mesh used in the Nek5000. For the real-life and industrial applications, it is necessary to employ scalable pre-processing tools for complex geometries. Together with WP 5 Task 5.1, an interface to optimized solutions of meshes with domain decomposition and load balancing should be created.

### 3.4.6 Load balancing

Nek5000 can obtain full scaling with uniform order on the petascale computations. When employ the strategy of adaptive refinement introduced in Section 3.4.1, the load balancing should be carefully considered due to the factor that the computations are different due to various order even on identify element.

### 3.4.7 Nek5000 roadmap to exascale

In summary, the development of the code is an essential new ingredient that will take the coupling idea from a theoretical concept to a practically useful tool for fluid flow investigations in exascale computations. Together with the other teams that develop algorithmic, modelling and performance analysis, all additions to the code will directly feedback into the code repository and are thus readily available to all users. The timetable is as follows.

#### *3.4.7.1 Document existing code architecture (Oct. 2011 – Mar. 2012)*

Documentation of all input parameters and setup files (.rea, .usr and SIZE). To create wiki pages with benchmarks of setup files for all users. Using DDT or other debug and performance tools developed WP3 Task 3.4 to learn and improve the code architecture. Assess refactoring possibilities.

#### *3.4.7.2 Implement error estimator and initial refinement code (Apr. 2012 – Sep. 2012)*

The goal is to develop new and improved computational tools for the study of flows in both fundamental and real-world environments. This includes implementation, assessment and adaptation of error estimators in nek5000 developed at KTH. Create an interface between the error estimator and Nek5000.

#### 3.4.7.3 Complete refinement development (Oct. 2012 – Mar. 2013)

To gain a fundamental understanding of most aspects of implementation of nek5000 with special attention to the large-scale simulation of incompressible flow.

#### 3.4.7.4 Implement load balancing using existing Nek5000 tool suite (Apr. 2013 – Sep. 2013)

To investigate and analyse the load balancing affected the refinement. Using the existing tools (Zoltan, Parmetis, Scotch etc.) to obtain full scaling and load balancing. To speed up Nek5000 with autotuning and performance analysis tools developed by WP3 Task 3.3.

#### 3.4.7.5 Undertake test and development on large scale applications (Oct. 2013 – Sep. 2014)

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. Such simulations are only possible with adaptive mesh refinement, proper boundary treatment and adapted post processing tools, all aspects to be developed during CRESTA.

## 3.5 IFS

The Integrated Forecasting System (IFS) is the production numerical weather forecast application at ECMWF. IFS comprise 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. While the focus is on the IFS model, it is expected that developments to the model should also improve the performance of the other IFS suites (EPS, 4D-Var and ENDA) mentioned above.

The resolution of the operational IFS model today is T1279L91 (1279 spectral waves and 91 levels in the atmosphere). For the IFS model, it is paramount that it completes a 10-day forecast in less than one hour so that forecast products can be delivered on time to ECMWF member states. The IFS model is expected to be increased in resolution over time as shown in Table 1.

| IFS model resolution | Envisaged Operational Implementation | Grid point spacing (km) | Time-step (seconds) |
|---|---|---|---|
| **T1279L91** | 2011 | 16 | 600 |
| **T2047L137** | 2014-2015 | 10 | 450 |
| **T3999LXXX** | 2020-2021 | 5 | 240 |
| **T7999LXXX** | 2025-2026 | 2.5 | 120 |

**Table 1 IFS model: current and planned model resolutions**

As can be seen in this table, the time-step reduces as the model resolution increases. In general halving the grid spacing increases the computational cost by 16, a doubling of cost for each of the 3 coordinate directions plus the time-step. However, in reality the cost can be greater than this, when some non-linear items are included such as the Legendre transforms and Fourier transforms.

It is clear from this that the IFS model from a computational viewpoint can utilize future supercomputers at Exascale and beyond. What is less clear is whether the IFS model can continue to run efficiently on such systems and continue to meet the operational target of one hour when running on 100,000 or more cores which it would have to do.

In a nutshell, IFS is a spectral, semi-implicit, semi-Lagrangian code, where data exists in 3 spaces, namely, grid-point, Fourier and spectral space. In a single time-step data is transposed between these spaces so that the respective grid-point, Fourier and spectral computations are independent over two of the three co-ordinate directions in each space. Fourier transforms are performed between grid-point and Fourier spaces, and Legendre transforms are performed between Fourier and spectral spaces. A full description of the above IFS parallelization scheme is contained in [6].

The performance of the IFS model has been well documented over the past 20 years, with many developments to improve performance, with more recent examples described in [7], [8], [9], [10], [11] and [12].

In recent years focus has turned to the cost of the Legendre transform, where the computational cost is O(N**3) for the global model, where N denotes the cut-off wave number in the triangular truncation of the spherical harmonics expansion. This has been addressed by a Fast Legendre Transform (FLT) development, where the computational cost is reduced to $C_L$*N**2*LOG(N) where $C_L$ is a constant and $C_L$<<N. The FLT algorithm is described in [13], [14], and [15]. While the cost of the Legendre transforms has been addressed, the associated TRMTOM and TRMTOL transpositions between Fourier and spectral space are relatively expensive at T3999 (>10 per cent of wall time). Today, these transpositions are implemented using efficient MPI_allgatherv collective calls in separate communicator groups, which can be considered the state of the art for MPI communications.

Within the CRESTA project we plan to address this performance issue by using Fortran90 coarrays to overlap these communications with the computation of the Legendre transforms, this being done per wave number within an OpenMP parallel region. If this approach is successful, it could pave the way for other areas in the IFS where similar communication can be overlapped with computation.

The semi-implicit semi-Lagrangian (SL) scheme in IFS allows the use of a relatively long time-step as compared with a Eulerian solution. This scheme involves the use of a halo of data from neighbouring MPI tasks which is needed to compute the departure-point and mid-point of the wind trajectory for each grid-point ('arrival' point) in a tasks partition (see [9] slides 19-29). While the communications in the SL scheme are relatively local the downside is that the location of the departure point is not known until run-time and therefore the IFS must assume a worst case geographic distance for the halo extent computed from a maximum assumed wind speed of 400 m/s and the time-step. Today, each task must perform MPI communications for this halo of data before the iterative scheme can execute to determine the departure-point and mid-point of the wind trajectory. This approach is clearly non-scaling as the same halo of data must be communicated, even if a task only has one grid-point (a rather extreme example).

To address this non-scaling issue, the SL scheme will be optimized to use Fortran90 coarrays to only get grid-columns from neighbouring tasks as and when they are required in the iterative scheme to compute the departure-point and mid-point of the trajectory.

In IFS the cost for computing Fourier transforms is $C_F$*$N_J$*LOG($N_J$), for each varying length latitude J=1.. N (N as above), where $C_F$ is a constant and $N_J$ is the number of grid points on latitude J. For optimal performance of the fourier transforms, full latitudes are statically load-balanced to tasks, where each task is responsible to computing FFTs for a subset of latitudes and a subset of atmospheric levels. The heuristic currently used will be reviewed as part of the CRESTA project and to explore an improved cost function for this load-balancing problem. The improved scheme should be applicable to all model resolutions.

Based on the above background description of IFS, we propose the following schedule of developments within the CRESTA project. It should be noted that some of these developments will overlap in time.

#### 3.5.1 Coarray kernel (4Q2011-1Q2012)

Develop kernel to investigate overlapping computation and communication using Fortran 2008 coarrays in an OpenMP parallel region.

#### 3.5.2 IFS CY37R3 port (1Q2012)

Port IFS model (CY37R3) to HECToR and analyse performance for model resolutions up to T2047.

#### 3.5.3 Exascale "Legendre transform" optimization (2Q-3Q2012)

The IFS transform library will be optimized to overlap the computation of the Legendre transforms with the associated communications.

These code developments will use the same strategy as prototyped in the Coarray Kernel, where the Legendre transform computation and associated coarray communications will execute in the same OpenMP parallel region.

This development will be tested using IFS model resolutions up to T2047 (a 2014-2015 sized model).

#### 3.5.4 IFS CY38R1 port (3Q2012)

Port IFS model (CY38R1) to HECToR. This code cycle is expected to become available in 2Q2012 and include support for the T3999 model resolution, FLT and substantially reduced memory requirements for computing the associated Legendre coefficients.

Run T3999 IFS model (an exascale case).

Assess "Legendre transform" optimization at T3999.

#### 3.5.5 Exascale "Semi-Lagrangian" optimization (4Q2012-2Q2013)

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.

#### 3.5.6 Optimization of Fourier latitude load-balancing heuristic (2013)

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.

#### 3.5.7 Development of a future solver for IFS (2014)

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

## 3.6 HemeLB

Based on the discussion in HemeLB's contribution to the WP6 needs analysis deliverable [CRESTA deliverable D62], the following are tools and libraries that need to be developed within CRESTA for HemeLB to deliver reliable exascale performance. As discussed in that deliverable, it is not sufficient that these libraries be delivered as research code capable only of use on specific platforms, each of these must be usable, manageable, deployable well-engineered, well-tested code.

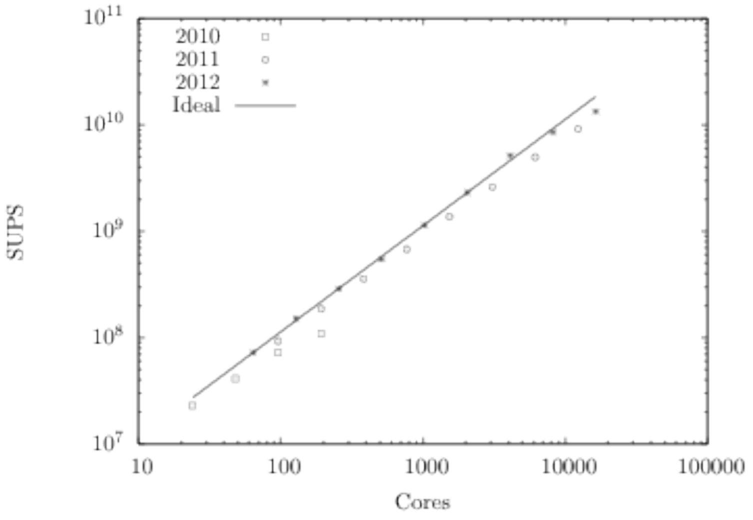

**Figure 1 HemeLB strong scaling performance. The 2010 data was before extensive redevelopment completed in 2011. The changes since (2012) have been focused on software engineering and improving code readability etc. We observe a small increase in performance despite more extensive use of object-oriented design.**

### 3.6.1 Visualisation and steering

Support for standard flow field visualisation for exascale simulations is a prerequisite for HemeLB to work at the exascale. As a first step, standard tools for flow visualisation, such as COVISE [17] will be linked to HemeLB in an ad-hoc fashion. However, to move forward, we will need to work with CRESTA collaborators to define a configuration system (API or DSL) so that visualisation tools can work with HemeLB's data *in-situ*, to support co-visualisation. In order to handle remote visualisation for steering at the exascale, data-volumes will need to be reduced by *in-situ* extraction of medically relevant properties, such as vessel wall stress, so that these smaller datasets can be shared. As HemeLB will form part of an ecosystem of computational physiology models within the Virtual Physiological Human, these systems will need to be made sufficiently configurable so that HemeLB results can be visualised alongside those of collaborating codes as part of a multiscale simulation.

### 3.6.2 Pre-processing

HemeLB uses the Parmetis [16] library to achieve domain decomposition for sparse geometries. Effort will be required within CRESTA to ensure this library scales appropriately. CRESTA enhanced or developed domain decomposition tools must support configurable interfaces for application specific domain decomposition. Later efforts will support continuous dynamic domain decomposition, in response to both simulation and system variability, including support for fault-tolerance.

### 3.6.3 Environments and operating systems

The vision of HemeLB as part of a clinically deployed exascale virtual physiological human will require usable environments for exascale deployment and job management. Job management infrastructure must support remote on-demand access from clinical settings, and appropriate algorithms for resource sharing must be developed for exascale hardware for this context. Operating system support for applications must be robust and easy-to-use, supporting multiple interacting applications using heterogeneous languages and paradigms for multiscale simulation.

Environmental support for auto-tuning of application configuration will be necessary, and this will require effort to support interaction with HemeLB's compile-time auto configuration facilities through CMake.

### 3.6.4 Introspection

HemeLB, as with many other applications, needs to be aware of its own progress as time passes. This application introspection, if it is not to be a blocker to exascale performance, will require attention from HemeLB developers and CRESTA tool effort. This will require not only performance measurement, but also support for report generation, visualisation of the correctness of the lattice-Boltzmann simulation. Within the multiscale VPH context, HemeLB introspection will need to interact with that of other applications. A clear API allowing application developers to discover on-going changes in the host environment, responding to faults and slow-downs, will be required for performance at Exascale.

## 4 Conclusion

In conclusion, the development of the CRESTA codes towards exascale is challenging. In principle, the task is not difficult, but for the scientifically most interesting cases the task seems almost impossible. The simple approach is to run the codes in much the same way as they are run today on terascale and repeat the setup a few 1000 times in an ensemble approach and call the result exascale. This approach has some benefits for cases were statistical averages and design optimization are important. Another approach is to simulate increasingly larger systems as the computing capacity grows. This is so called weak scaling. This is motivated and important for e.g. IFS in which case resolution must be improved in order to make more accurate predictions. For most cases, however, increasing system size adds only marginal scientific value. From a scientific point of view the most interesting is strong scaling. This means that small systems should be simulated over longer times. This would be important for e.g. GROMACS. Important biomolecular mechanism may take 10s of seconds in reality, which using the algorithms of today could take 10-100 years of computing.

## 5 References

[1] OpenFOAM web site, See: http://www.openfoam.org/download/

[2] Extend project web site. See: http://www.extend-project.de/the-extend-project

[3] Chapman D.: *Computational Aerodynamics Development and Outlook*, AIAA Journal, Vol. 17, No.12, pp. 1293-1313 (1979).

[4] Fröhlich J.: *Large Eddy Simulation turbulenter Strömungen*, Teubner Verlag, 1. Auflage (2006).

[5] ERCOFTAC web site. See: http://www.ercoftac.org/

[6] Barros, S. R. M., Dent, D., Isaksen, L., Robinson, G., Mozdzynski, G. and Wollenweber, F., "The IFS Model: A parallel production weather code", Parallel Computing 21 1621-1638 (1995).

[7] Hamrud, M., "Report from IFS scalability project", Technical Memorandum 616, (2010). See: http://www.ecmwf.int/publications/library/ecpublications/_pdf/tm/601700/tm616_rev.pdf

[8] Salmond, D. and Hamrud, M., "IFS scalability and computational efficiency", (2010). See: http://www.ecmwf.int/newsevents/meetings/workshops/2010/high_performance_computing_14th/presentations/Salmond_Hamrud.pdf

[9] Mozdzynski, G., "IFS: RAPS11 and model scaling", (2010). See: http://www.ecmwf.int/newsevents/meetings/workshops/2010/high_performance_computing_14th/presentations/Mozdzynski_scaling.pdf

[10] Salmond, D., "IFS performance on the new IBM Power6 systems at ECMWF", (2008), See: http://www.ecmwf.int/newsevents/meetings/workshops/2008/high_performance_computing_13th/presentations/Salmond.pdf

[11] Mozdzynski, G., "IFS scaling", (2008), See: http://www.ecmwf.int/newsevents/meetings/workshops/2008/high_performance_computing_13th/presentations/Mozdzynski.pdf

[12] Mozdzynski, G., " A new partitioning approach for ECMWF's Integrated Forecasting System (IFS)", (2006), See: http://www.ecmwf.int/newsevents/meetings/workshops/2006/high_performance_computing-12th/pdf/George_Mozdzynski.pdf

[13] Rokhlin, V. and Tygert, M., "Fast algorithms for spherical harmonic expansions," SIAM Journal on Scientific Computing, 27 (6): 1903-1928 (2006). See: http://cims.nyu.edu/~tygert/sph2.pdf

[14] Tygert, M., "Fast algorithms for spherical harmonic expansions, II," Journal of Computational Physics, 227 (8): 4260-4279 (2008). See: http://cims.nyu.edu/~tygert/spharmonic.pdf

[15] Tygert, M., "Fast algorithms for spherical harmonic expansions, III," Journal of Computational Physics, 229 (18): 6181-6192 (2010). See: http://cims.nyu.edu/~tygert/butterfly.pdf

[16] Schloegel, K., Karypis, G., & Kumar, V. (2002). Parallel static and dynamic multi-constraint graph partitioning. Concurrency and Computation: Practice and Experience, 14(3), 219–240. doi:10.1002/cpe.605

[17] COVISE web page. See: http://www.hlrs.de/organization/av/vis/covise/

[18] Mazzeo, M. D., & Coveney, P. V., HemeLB: A high performance parallel lattice-Boltzmann code for large-scale fluid flow in complex geometries.

Comput. Phys. Commun., 178(12), 894–914 (2008). doi:10.1016/j.cpc.2008.02.013.

[19] Mazzeo, M. D., Manos, S., & Coveney, P. V., In situ ray tracing and computational steering for interactive blood flow simulation. Comput. Phys. Commun., 181, 355–370 (2010). doi:10.1016/j.cpc.2009.10.013

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

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