D6.2 – Needs analysis

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:	01/10/2011
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), Artur Signell (ABO), Jan Westerholm (ABO)
Reviewer(s)	Christian Wagner (DLR), Harvey Richardson (Cray)

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	17/01/2012	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	
		NEK5000 added	
		HemeLB added	
1.0	1/3/2012	Final corrections made to document based on reviewer feedback	J.A. Åström (CSC)

Table of Contents

1 Executive Summary

This 'Needs analysis' contains an overview of what the CRESTA applications codes need in order to be developed towards the exaflop/second realm. The summaries of the requirements for the co-design codes are as follows:

ELMFIRE: would benefit from real-time visualisation, automated hardware failure recovery, an exascale linear solver and file I/O, exascale profiling and debugging. These aspects are being dealt with together with other work packages within the project.

GROMACS: The main challenge for Gromacs is to improve scaling for lattice summation electrostatic interactions through more efficient FFT libraries, or by using completely different algorithms that do not involve all-to-all communication. For the remaining part of the code, the requirements are more focused on improving parallelism on all levels, including tasks over CPUs/cores/GPUs, and improving IO data handling for extremely large systems. For small systems, the exascale needs will have to be addressed with ensemble techniques that do not require synchronization between all nodes for each time step.

OpenFOAM: OpenFOAM® is considered to scale well to thousands of cores. Some important use cases need the OpenFOAM®-extend versions including the GGI (General Grid Interface). The possible bottleneck GGI seems to work well after maintenances had been done few months ago from the OpenFOAM®-extend community. OpenFOAM®-extend, however, still has to be prepared for the exascale architectures.

NEK5000: needs an integration of the pre- and post-processing codes, a p-type mesh refinement, hybrid parallelism, parallel file I/O, and an investigation of the exascale performance of BLAS, among other things. These issues are developed with the other WPs.

IFS: needs to take advantage of Fortran co-arrays. In order for proper exascale implementation the code needs profiling and debugging tools that can handle this type of data-structures. Furthermore, Fourier transforms and multi-grid will be developed in cooperation with other WPs.

HEMELB: would benefit from e.g. exascale visualization, debugging and profiling tools, failure tolerant MPI and dynamic domain decomposition. These issues will be investigated in cooperation with other WPs.

2 Introduction

This document contains a 'needs analysis' over the actions needed to develop the CRESTA codes towards exascale performance. The needs of the different codes are presented in chapter 3. The codes and their purpose are as follows:

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 the University of Stuttgart will be on a specialised code for turbo machinery. The objective is to simulate a whole hydraulic machine on exascale architectures.

ELMFIRE: is a gyro-kinetic particle-in-cell code that simulates movement and interaction between particles moving at high speed in a torus-shaped geometry on a three-dimensional grid 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 these kinds 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.

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
CPU	Central processing unit
PETSC	A computer code algorithm library
GPU	Graphics processing unit
BLAS	Basic linear algebra library
FFT	Fast Fourier transform
O(N)	Order - N
LES	Large-Eddie simulation
GGI	General grid interface
O(N)	Order - N
API	Application programming interface
HDF	Hierarchical data format
LB	Lattice-Boltzmann

3 Needs analysis for the codes

3.1 ELMFIRE

ELMFIRE lacks real time visualization support, which would bring the usability of the program to a new level. A full run of the program must be completed before the results can be analysed and visualized. Real time visualization would be beneficial not only to be able to partly analyse the results before the run has finished but also to detect problems with the program or the input data before wasting a week of CPU time. This will be done in collaboration with WP5.

Check pointing is done in ELMFIRE at given intervals (every Nth time step). Check pointing is performed by dumping all particle data to files, which can be read back later to initialize the program. Around 72 bytes per particle are stored, meaning a checkpoint of an ITER simulation would require around 40TB. It would be beneficial to be able to remove the custom code for this from the program and have a more efficient approach for fault tolerance or fault recovery. What we would like to see is an automated approach for recovering from hardware faults by not stopping the program completely but instead revert to the latest checkpoint, bring in new processors if needed, and continue running from the checkpoint.

ELMFIRE constructs a sparse, linear, modified gyro kinetic Poisson equation that is solved in parallel using PETSc. It is at the moment unclear if PETSc scales to exascale problems or if this needs to be replaced by another approach. Research on this topic will be done in collaboration with WP4.

File output in ELMFIRE is mostly done by all processors sending their data to processor 0, which then writes the data to disk. Some initial prototyping has been done for moving to a HDF5 based output format but falls out of the scope of what we are able to do during the project. This could be done in cooperation with work package 5.

Running an ITER simulation on ELMFIRE would involve hundreds of thousands of cores. Profiling of such a large application would likely be difficult using tools of today. What would be needed is separate numbers for CPU usage, MPI/network traffic, load balancing information and file I/O for all processes and good ways of visualizing the results to be able to detect where possible bottlenecks are. We hope that we can work together with WP3 to ensure this is made possible.

Debugging is a topic of interest, mainly in the scope of being able to find out why the run with 100K processes crashed at some point. Ideally we would like to see a way that debugging could always be enabled for a program but not really kick in before there is a problem. The overhead of debugging should be minimal and there shouldn't be a need to re-compile and re-run the program using 100K cores to get debug info. If an optimized program takes days to crash using 100K cores it will take even longer if running in debug mode. It is not really feasible to re-run the program to find out why it crashed.

3.2 Gromacs

Gromacs is a major open source code that performs classical molecular dynamics simulations based in 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 inhomogeneities 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 [16].

The central challenge for pushing Gromacs to exascale performance is to develop algorithms and problem formulations that can make efficient use of this amount of floating-point instructions. While there is important work to be done on classical parallelization improvement, it will not be possible to achieve this solely by increased parallelization that reduces the wallclock time per iteration, since this would bring the time-step in the nanosecond range for small systems, which is completely out of range of current architectures.

3.2.1 Performance & scaling of the core MD algorithm

Gromacs is parallelized on several different levels, which is critical to reach high application performance rather than merely good relative scaling. On the lowest level, we have long employed Single-Instruction Multiple-Data parallelism on modern hardware, but largely with hand-written assembly language. One important need is to formulate these algorithms in a more portable fashion that makes it possible to rapidly recompile and adapt the computational kernels to new generations of hardware with different capabilities. It is also important to include better support for streaming co-processors such as GPUs and/or heterogeneous parallelization.

On a higher level, Gromacs uses message-passing parallelization to distribute work over separate nodes. While this works well, it is currently the main bottleneck for scaling. For systems that only contain simple interactions, Gromacs scales well down to roughly 250 atoms per core (i.e., weak scaling is essentially perfect). However, most biomolecular simulations involve electrostatic interactions, and these have to be treated with lattice-summation based algorithms involving fast Fourier transforms. While there are parallel FFT implementations, the dimensions of the grid are relatively small, and this means the latency of the FFT implementation becomes a bottleneck. Lower-latency FFTs would go a long way towards improving this problem, but it might also be possible to develop completely new algorithms that do not depend on FFTs to solve long-range electrostatic interactions, for instance multi-grid solvers or even fast multiple algorithms. Work on FFT libraries in particular will be performed in WP4, and also as a CRESTA co-design team.

After the long-range electrostatics, the next current limitation is the latency and communication patterns for the domain decomposition. This is likely not possible to solve in any other way than hard manual work in the core of the code, and this has to be pursued by the Gromacs team in the present project. However, this communication in turn relies on the MPI communication library. We have already seen that other codes (such as NAMD) have achieved improved performance by directly targeting Infiniband OpenFabrics verbs (http://www.openfabrics.org) or other communication libraries closer to hardware, and such implementations could also be highly useful for Gromacs. In general, CRESTA WP2 should be able to help find lower-latency communication strategies for Gromacs.

Finally, present hardware with ever-increasing core-counts opens the possibility to introduce an intermediate parallelization layer by directly utilizing the shared memory inside each compute node. This can be surprisingly hard with non-uniform memory architectures (NUMA). It might be possible to address better with a task-based parallelization to avoid synchronization, and here Gromacs would benefit significantly from better libraries to automatically distribute tasks over resources based on memory

latency e.g. between different cores, dies, and what co-processor is attached to what bus and what CPU die.

In addition, the development is highly dependent on availability of good profilers that simultaneously work on multi-thread and multi-MPI-job level to find data imbalances and points where we are waiting for communication.

3.2.2 Extreme system parallelism and IO needs

The obvious way to utilize exascale resources is to increase the simulated system size. Typical targets for the US Blue Waters system include biomolecular systems with over 300 million particles. For these systems, Gromacs has a current major bottleneck in how it handles IO for simulation data. Currently, the system input has to be read into the master node and then distributed, which takes a very long time and it can even be impossible for large systems running on hardware without any swap space available. In the same way, the current algorithms for checkpointing and writing trajectory data rely on sending the data through the master node. Here, we have a clear need for a good, portable, and small C/C++ library that can handle distributed data in a rudimentary fashion, in particular for IO. A problem with solutions like PGAS languages is that they are less portable, and we do not want to include a completely different parallelization approach just to handle IO for the very largest simulations.

3.2.3 Scaling for small biomolecular systems

One significant failure with most current parallelization work for biomolecular system is that it is frequently focused on using as many cores as possible by increasing the problem size. While this provides great scaling plots, it is of very limited use to practical life science applications where the vast majority of molecular systems studied are well below a million atoms, and frequently down to 30,000 atoms. To utilize exascale resources for these applications, we need a completely different approach based on running huge ensembles of closely coupled simulations [17]. Some challenges here revolve around the algorithms used for sampling, but there is also a great need for better tools for checkpointing automatically and reissuing failed tasks, and not least: data handling. In particular, Gromacs would benefit from better libraries to place processes on the hardware in large clusters as well as how to place tasks/threads within nodes to minimize the communication latency and bandwidth. We also see a need for better abilities to start & stop sub-tasks in an ensemble of simulations, e.g. when hardware problems are detected, without separating a gigantic ensemble into thousands of separate cluster jobs that would not be able to communicate. This will be addressed in WP6.

3.3 OpenFOAM

OpenFOAM® is an open source code, and multiple versions are in common use. For the purposes of this project, we have chosen to consider two widely used versions, namely

- The OpenFOAM foundation release from OpenCFD Ltd (a company which owns the OpenFOAM trademark)[1]
- The release from the OpenFOAM Extend Project[2].

Whilst there are several potential challenges in running OpenFOAM at the exascale, the most challenging aspect of preparing OpenFOAM for exascale systems remains the parallel scaling of the computationally intensive sections of the code.

OpenFOAM, in general, is considered to scale well on current systems. For simple use-cases it has been shown to scale to thousands of processors[3]. It is likely, however, that exascale machines will have hundreds of thousands of processors. Further work is required to understand what the limiting factors will be as the code is scaled up, but it is expected that the performance will be limited by inter-process communication, and in particular, large reduction operations[4]. On a single-core basis, the performance is probably limited by memory latency[4].

Mesh creation and partitioning are also very important components of the code, and it must be ensured that these will scale up as well as the main solvers. Recent versions of OpenFOAM use the PT-Scotch library [5] to do mesh creation and partitioning in parallel. We will work with partners in WP4 and WP5 to investigate the limitations of these pre-processing steps, both in terms of their algorithms and their interaction with the solver part of the code.

To determine the needs of the OpenFOAM application, the project will consider a set of specific use cases. The application's needs described here are based on these use-cases. As one of the use-cases, we have considered the flow of air around a moving motorbike. This is one of the tutorial examples distributed with OpenFOAM and has been identified as being both representative of large problems, and having characteristics that could lead to non-trivial issues when scaling to very large numbers of processors. In order to help stimulate the development of OpenFOAM and to motivate improvements to the code, two specific cases have been prepared by the Institute of Fluid Mechanics and Hydraulic Machinery (IHS) of the University of Stuttgart. The first one is a quite simple geometry of a square cylinder that is also an ERCOFTAC test case in order to validate LES. This case should be suitable for first computations in order to validate as simple as possible the new implementations. The second one is a real hydraulic machine that is more complex geometry but is the most important case for the Institute, because of the complexity of moving parts in particular.

Important use-cases at the Institute of Fluid Mechanics and Hydraulic Machinery (IHS) of the University of Stuttgart are simulations of a whole hydraulic machine that have particular needs, in particular the need to simulate moving parts. This is one of the main advantages to the OpenFOAM Extend version. In general, the same tools for pre- and post processing could be used in all OpenFOAM versions. In order to simulate a whole hydraulic machine, however, a moving mesh interface like GGI is needed. That means there is a need to check that GGI works in order to simulate rotor/stator interaction with for example the OpenFOAM-2.1 version. Assuming it can be used, GGI should then be checked for scalability because the experience shows that GGI could probably be bottleneck on Large Scale Computing.

The relevant test case is the use of OpenFOAM to simulate the flow in an entire hydraulic machine using 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. This simulation therefore requires very fine computational grids and consequently a very high computational effort.

The flow in a hydraulic machine has relative high Reynolds numbers (Highly Turbulent Flow) in a test rig size of about $3*10^6$ – $5*10^6$. The number of vertices and the small time step size will lead to a requirement of 80 millions core hours to get a full converged simulation. This would lead to a usage of 50000 cores for about 65 days.

The resulting data size of a full solution is about 250GB for one time step. To get transient data of global parameters, like torque, efficiency or flow fields in selected points or lines, the storage needs about 100GB. The number of time steps that have to be saved, depends on the visualisation of the instantaneous flow phenomena. E.g. storing every degree runner rotation for one complete rotation would require 360*250GB=90000GB=90TB.

3.3.1 Scalability of main computation

A traditional approach to optimising OpenFOAM will be adopted, namely to iteratively profile the code, identify bottlenecks and seek to improve the scalability of those parts of the code which do not scale well. To do this effectively, we will need to understand the expected trends for the balance between compute, communication, memory, and I/O in future hardware so that we are optimising for the platforms of tomorrow, as well as those on which the code is currently being run. We would work with WP2 to understand the implications of these trends on the future performance of OpenFOAM, and as we scale up the problems being looked at we are very likely to benefit from the

expertise of partners in WP2 and WP3 in the use of performance analysis and debugging tools for exascale problems.

3.3.2 Pre- and postprocessing

Pre- and Post-processing are two important tasks in Computational Fluid Dynamics. In case of a LES these tasks are time expensive. The Institute of Fluid Dynamics and Hydraulic Machinery (University of Stuttgart, Germany) works on the OpenFOAM® version OpenFOAM-1.6-extend. The tools for Pre- and Post processing should be the same to the tools of other OpenFOAM® versions.

3.3.3 Preprocessing

For Preprocessing a block structured mesh is created and written out as OpenFOAM® input. The next step is to refine that mesh, in order to reach the appropriate number of computational domain vertices as mentioned above, see the schematical view in fig 1. This could be done with the OpenFOAM® utility *refineMesh* which refines the mesh in every direction.

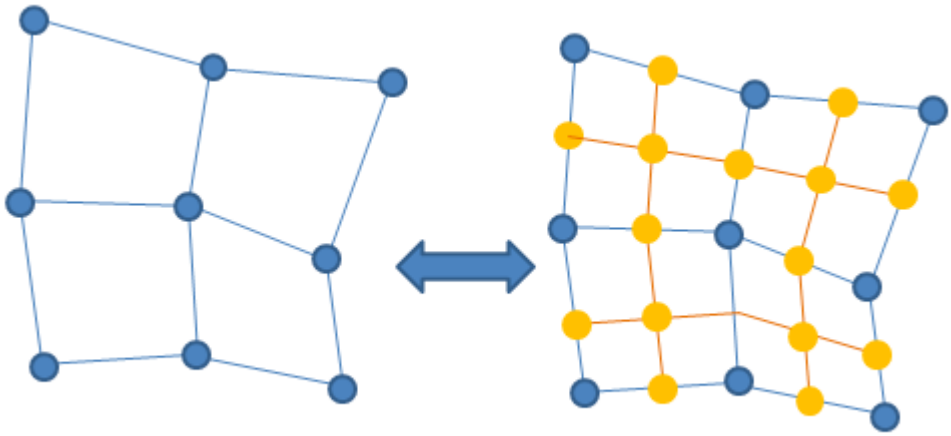

Fig. 1: Schematical view of the grid refinement process

After the refinement, domain decomposition is needed. This can be realized with the *redistributeMeshPar* that is coupled to the PT-Scotch library [5] to mesh partitioning in parallel. This enables cooperation with partners in WP4 and WP5 to investigate limitations in these pre-processing steps. Ideally, the mesh refinement and the domain decomposition should be done into a single step.

Furthermore, another topic to realise a LES, is the need of a very good initialisation. In order to do this, the traditional way is to get a first solution of the problem with a standard turbulence model of the Average Navier-Stokes Simulation (RANS) family, in example k-epsilon model or the SST-k-omega model. The Reynolds Average Navier-Stokes Simulation is realizable on coarse computational domain grids and the results are then mapped on the grid prepared for the LES. There exists a tool in OpenFOAM® called *mapFields* for doing that, but to our knowledge the tool runs not in parallel. This enables cooperation to parallelization group of the WP3 in order to find out an efficient parallel programming model to parallelize *mapFields*.

3.3.4 Postprocessing

The visualization tools Paraview and Covise (from HLRS) are currently being used successfully for postprocessing. The amount of data, typically 90TB, is difficult to transfer from the HPC system to do postprocessing locally to the user. Thus, parallel postprocessing on up to 50000 cores is needed. Furthermore, efficient data reduction and compression of the results is needed in order to reduce I/O time of the visualization tools. We will need to work with the colleagues from the visualization group in WP5 on that topic in order to see if it is possible and if not we should find another process to visualise the results.

The general intended workflow is shown in fig. 2. It consists of

- Grid generation
- Decomposition and refinement
- Simulation
- Data reduction and compression and
- Visualisation.

It is essential, that all tools, which work on the refined mesh, are running highly efficiently on many thousands of cores, in order to obtain an overall good performance for the simulation.

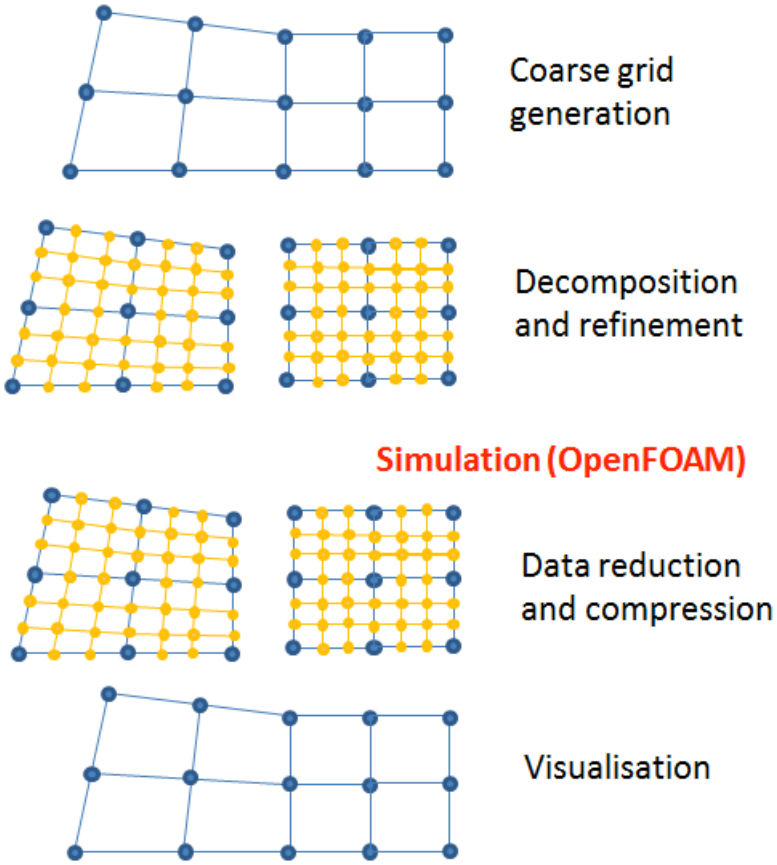

Fig. 2: Schematical view of the overall work process

3.3.5 Fault tolerance

OpenFOAM, like nearly every existing massively parallel code has no explicit functionality (other than checkpointing) to tolerate faults in the underlying system. We plan to further investigate the parallelism patterns used in the code and determine the scope for making modifications that could work with system-level fault tolerance mechanisms. We will need to work with WP5 in order to understand the form that such mechanisms are likely to take.

3.4 NEK5000

Nek5000 is an open-source code for the simulation of incompressible flow in complex geometries. Nek5000 already scales up to 200,000 processors and our work in CRESTA will focus on extending scalability on exascale supercomputers. Towards exascale scalability new theoretical solutions for parallelism will be implemented. These solutions include adaptive refinements, alternative discretisation and hybrid parallelisation. Extra care will be taken to data handling, load balancing and pre and post processing. In order to adapt the Nek5000 code for the exascale computation, we

will work in close collaboration with other CRESTA work packages throughout the entire development process.

The developments of Nek5000 in the CRESTA project will focus on two main points. First, we will develop and implement software interfaces between Nek5000 and the codes responsible for mesh generation and visualization in the pre- and post-processing stages. This will allow us to capture the features and complexity of geometries as well as predict the errors in the mean flow field. This work will be done collaboration with WP5. Second, we will work on developing and implementing a p-type adaptive refinement in Nek5000. In collaboration with Nek5000 main developer Paul Fischer at Argonne National Laboratory, we will analyse the stability conditions for the various order levels, and we will implement this new approach in the Nek5000.

In addition, other Nek5000 optimizations for exascale computing platforms as discussed in the project plan will be considered (alternate discretization, hybrid etc.).

The scalability of Nek5000 mainly depends on the global communication due to the pressure constraint. Nek5000 employs the Crystal Router algorithm to implement the global communication. This is an efficient (but rather old) technique for collective communication with massively parallel processors connected in a hyper-cube topology. The algorithm collects many small messages in an effort to reduce the latency costs dominate. We will use the performance analysis tools and data collection developed by WP3 to analyse and improve the algorithm.

The execution time of Nek5000 is dominated by the calculations of small dense, rectangular matrix-matrix and matrix-vector products. On the majority of the computer architectures, Nek5000 uses the third-party library Basic Linear Algebra Subroutine (BLAS) to carry out these calculations. The BLAS was original designed for the calculations of large dense matrices. To speed up the Nek5000 code we will work together with WP4 to analyse the BLAS limitations and improve the other existing libraries for the exascale computations. We will also work with WP3 to employ the auto-tuning techniques and exascale compilers.

In the present state, Nek5000 does not employ any hybrid approach to parallelization. All communication is handled by MPI, which has been proven to be very efficient, mainly due to the element structure of the mesh. However, a hybrid approach with MPI/OpenMP might be considered in the future. The analysis of advantages and disadvantages of a hybrid approach in Nek5000 will be carried out with WP3.

Also, with the help of the Cray Experts, we will analyse the file I/O requirements of the code (which might be quite considerable depending on the flow case), and optimize using parallel techniques. So far, the code allows writing from a user-defined number of nodes, however creating separate files. Using either HDF of MPI-I/O we might further optimize/simplify the file handling, in particular for restart/checkpointing using a multistep time-integration method.

3.5 IFS

The main developments to the IFS model to get it to run efficiently on future exascale systems will be done by ECMWF. These developments require the use of Fortran90 coarrays to optimize the communications associated with the Legendre transforms and to improve the scalability of semi-Langrangian communications that are described in some detail in D6.1.1.

The availability of Fortran90 coarrays is paramount for these developments and further it is essential for both developments that coarray transfers between images be supported within the context of OpenMP parallel regions.

These capabilities have already been tested using a “Coarray Kernel” on HECTOR using the crayftn compiler, however, coarray transfers only work today on HECTOR if they are constrained within an OpenMP critical section. ECMWF will work with partners WP3 to find a resolution to this issue.

It is equally important that performance analysis tools and debuggers work reliably for large production applications such as IFS, so ECMWF is keen to test such tools in WP3 during the development cycle and to provide feedback to the tool developers. These tools need to support Fortran90 coarrays and particularly when they are used within the context of OpenMP parallel regions.

ECMWF will work with WP4 to optimize the Fourier latitude load-balancing heuristic used in IFS, to improve the scalability of the Fourier transforms.

Finally, ECMWF will also work with WP4 to investigate a new multi-grid solver for extreme scaling. This work would involve comparing the existing solver used in IFS with a potential replacement, and tested in a much simpler code such as a shallow water model. It should be noted that a new replacement solver is not part of ECMWF's current research plans and should be considered more speculative.

3.6 HemeLB

The development work on HemeLB will be mostly undertaken by UCL. The work described in sections 3.6.2 and 3.6.5 below will be informed by co-design with other partners in WP5.

3.6.1 Engineering requirements for software components

Scientific codes addressing problems that challenge exascale resources necessarily address very complex problems. Challenges from problem size and data volume are strongly correlated with software engineering challenges of code complexity and information management. HemeLB has a strong object oriented design, necessitated by the complexity of the problems we try to address.

Software components developed within CRESTA for use by co-design vehicles must, of course, scale in terms of computational resources. However, for this work to have utility for co-design applications, they must also scale in terms of problem complexity.

Therefore, for utility at the exascale, tools must be well architected, modular, well tested, have clearly defined dependencies and be deployable in widely heterogeneous environments. Tools that provide exascale performance that are not engineered to be usable, sustainable, and manageable within the context of exascale scientific applications cannot be considered to be successful. The "definition of done" for such work, must, therefore, reflect these requirements.

Efforts toward exascale computing cannot neglect maintainability or development and integration time in favour of optimum use of resources. Tools must integrate easily with scientific applications and deploy smoothly to a wide variety of target machines. Interactivity and usability for scientific insight must be valued as much as efficient computation. At the exascale, computation can be assumed to be an abundant resource, relative to the time and effort spent developing, managing, and coordinating the development of scientific codes.

3.6.2 Visualisation

High-quality, medically relevant visualisation is a core aim of the HemeLB project. By providing clear spatial representations of hemodynamics in complex vascular topologies, HemeLB will enhance and support clinical decision-making.

Some of the visualisation needs for HemeLB, being based around the visualisation of flow fields, match those of other computational fluid dynamics visualisation problems. Achieving efficient visualisation of exascale flow-fields is a complex problem in its own right, and HemeLB will benefit, along with other CRESTA co-design vehicles, in advances in this area. Scalable support for visualisation

techniques such as dynamic line integral convolution and volume rendering will be of value.

Given the volumes of data and the complexity of the simulations involved, it is likely that co-visualisation will be necessary - that is, production of visualisation data on the same computational infrastructure as the simulation. This could either be fully *in-situ* visualisation (with the same process carrying out both visualisation and simulation) or within a separate process on the same infrastructure. Support for asymmetric parallelism may be important here, with visualisation taking place on one or more cores of a node, with the other cores carrying out simulation, and communication between simulation and visualisation taking place through shared memory.

As a lattice Boltzmann simulation, however, there are ways in which visualisation support for HemeLB differs from that required for traditional CFD.

The memory layout of the lattice-boltzmann field (from which the flow-field may be obtained by a weighted averaging), optimised for computation, may be arranged in ways unusual for interpretation by CFD visualisation codes. If the visualisation library is to be a pluggable software component, it is essential that it be capable of adapting to a variety of efficient methods of communication with simulation code. For optimum efficiency we suggest this could be through a domain specific language capable of specifying memory layout for shared-memory communication in an *in-situ* visualisation context.

Although lattice Boltzmann simulations have gained traction due to their excellent scaling properties, best practice for its application to scientific problems remains an area of active research. Issues such as the choice of parameters needed to produce a pseudo-incompressible lattice Boltzmann fluid, and the appropriate selection of boundary conditions to model convoluted or deformable surfaces mean that application of LB techniques can be a subtle process. For this reason visualisation of the behaviour of the LB implementation, at a level below that of the intended physical observables, will also be important.

HemeLB's targeted deployment into clinical environments also differentiates it from other CFD visualisation tasks, by placing the emphasis on visualisations that resonate with the intuition of medical practitioners, as opposed to engineers or physical scientists.

Extraction and visualisation of clinically relevant properties, such as the stress on the vascular walls which can lead to rupture or induce malformation, is therefore of paramount importance for HemeLB. As these properties can be peculiar to the application at hand, it is therefore necessary that visualisation libraries developed for the exascale are componentised and made configurable in such a way as to support re-use for novel, domain-specific visualisations. This may include visualisations that attempt to mimic the outputs of experimental observations of modelled systems, so as to better resonate with clinical intuition.

Our target for HemeLB is deployment into clinical contexts. Thus, interfaces developed to interact with and steer simulations must take into account the need for remote interaction between exascale computing resources and practitioners in hospitals. Job management infrastructure on exascale resources must support resource allocation strategies appropriate to use in clinical practice.

3.6.3 Introspection

Applications must carry out introspection -- to be aware, as they run, of how they are running, and report this to developers and users. Tools to support this activity for exascale resources are needed, as introspection can be a block to scalability, and yet introspection is vital for code development.

Information that must be collected includes on-going timing measurement and profiling, numerical stability, simulation progress and logging, debug information, and the impact of adjustments for fault tolerance. Reporting objectives include monitoring of running simulations, including potential for adjustment via steerability to rescue failing simulations, archiving of textual and machine-readable reports, and visualisation.

3.6.4 Lattice Boltzmann

The family of lattice-Boltzmann methods have, over the last two decades, been used for studying a wide variety of flow problems. One of its strongest "selling points" is the relative straight-forwardness of designing an efficient parallel implementation, due to the method requiring each lattice point to only communicate with its neighbours, in order to advance the simulation state forward one time-step. Most parallel codes show near-perfect weak scaling up to thousands of cores, with state-of-the-art implementations scaling up to hundreds of thousands (for example LUDWIG, LB3D [6]). These examples, however, are for very simple geometries, i.e. solid cuboids of fluid, which can be easily decomposed across nodes and have the communications mapped onto the network fabric's topology. (Clearly, this is predicated on nodes being reliable.)

However the bottleneck for simulation performance for lattice-Boltzmann is usually attributed to the relative slowness of main memory, whether memory bandwidth [10][7] memory latency or the smallness of the translation lookaside buffer (TLB, used to translate virtual memory addresses to hardware addresses) requiring extra loads from main memory[11] This issue will have to be kept in mind at all points during CRESTA. Due to the complex, platform-dependent interplay of factors, we see some form of auto-tuning as a key approach to reaching high per-node performance. This library must interact efficiently with existing standard build-management tools, such as CMake [12] to allow a fluent interface to configurable builds of client tools such as HemeLB.

No fault tolerance is currently implemented within HemeLB. We observe that MPI implementations are typically not fault-tolerant and this must be addressed. If the application, through introspection or notification by the operating system, is notified of an impending failure, then the data for the sub-domain could be transferred to a hot spare, its neighbouring tasks notified of the change and the simulation could proceed. This would require only a modest application programming effort, but requires over-provision of nodes. Dynamic load balancing (discussed below) and/or repartitioning would clearly reduce the need for over-provision, by allowing the simulation to proceed at slightly degraded performance in the case of node failure.

3.6.5 Geometry and pre-processing

In contrast to most LB applications mentioned above, HemeLB is optimised for sparse geometries such as the vasculature, where the typical fluid fraction of a cuboidal bounding box is ~5%. This requires that we be able to distribute these fluid sites across cores in a more flexible manner than a simple Cartesian decomposition. Currently we are using ParMETIS, a parallel graph-partitioning library [9], to produce a static decomposition at simulation start up. (This same choice has been made by the developers of MUPHY for comparable problems [8].)

As our systems get larger, the demands placed on the decomposition algorithm will grow.

The domain decomposition also affects the performance of the visualisation subsystem. Ideally this should be taken into account by the decomposition algorithm, in order to co-optimise the whole simulation.

Dynamically changing the domain decomposition during the simulation, either in response to variation in the workload or node availability, is a theoretically attractive option. This has been implemented by, for example, the Charm++ project, used by the NAMD molecular dynamics code and would delegate much of the work of domain decomposition to the API and run-time system. While this might significantly reduce the code complexity of HemeLB it would require significant development effort and is unlikely to be completed within the timescale of CRESTA. However it would require a large redevelopment of HemeLB.

HemeLB currently used MPI-IO for reading its geometry data and writing snapshots of the simulation state. The geometry files are block-decomposed and each task reads only the (variable-length) blocks for which it is assigned responsibility. The snapshot files are currently a simple custom binary format, written with collective MPI-IO functions. We expect to replace this with a more sophisticated format through the use of a library, such as HDF.

Our pre-processing tool is currently a workstation level application, that uses VTK [13] to convert a description of the surfaces of the vasculature into our custom format. This is currently only practical for systems with less than approximately 10^8 lattice points, at least two orders of magnitude below an exascale system. We will consider how this can be parallelised across a smaller cluster.

3.6.6 Multiscale

HemeLB forms part of wider efforts in computational physiology. HemeLB will interact with other instances of HemeLB running at different scales, with simpler one-dimensional models of the remainder of the circulatory system, and with models of surrounding solid tissue. HemeLB's involvement with the Virtual Physiological Human (VPH) [14] and Multiscale Applications on European e-Infrastructures (MAPPER) [15] projects is a major part of this effort.

Tools to support multiscale modelling, (i.e. to support interaction between different programs), on exascale resources will therefore be important for HemeLB. Visualisation, decomposition, and introspection tools must have APIs generic enough to work not just with HemeLB, but also with a heterogeneous variety of application codes, so that multiple codes may be executed, visualised and evaluated together.

Different components of a multiscale simulation may exist on distinct, spatially separated exascale resources, so support for wide-area exascale computing will be necessary. Exascale operating systems must support fluent deployment of multiple codes using a variety of languages, paradigms, and approaches.

4 Conclusion

In conclusion, exascale will set a lot of challenges on data handling like pre- and post-processing, visualization, meshing, etc. These challenges consist in the vast amount of data to handle, but also the usefulness of visually inspecting the data. Furthermore profiling tools and debuggers will have to cope with an enormous amount of tasks. Existing libraries and algorithms will also face new types of challenges. Finally, the 'fault-tolerance' problem is still largely unresolved including a realistic estimate of how severe it may become.

5 References

[1] OpenFOAM® - The open source CFD toolbox, See: http://www.openfoam.org/

[2] The OpenFOAM® Extend Project, See: http://www.extend-project.de/

[3] G. Pringle, "Porting OpenFOAM to HECToR - A dCSE Project", Technical Report (2010), See: http://www.hector.ac.uk/cse/distributedcse/reports/openfoam/openfoam.pdf

[4] G. Wierink, OpenCFD Ltd., Personal correspondence

[5] Scotch Home Page, See: http://www.labri.fr/perso/pelegrin/scotch/

[6] Bernd, M., & Frings, W. (Eds.). (2011). Jülich Blue Gene/P Extreme Scaling Workshop 2011. (M. Bernd & W. Frings, Eds.). Retrieved from http://www2.fz-juelich.de/jsc/docs/autoren2011/mohr1/

[7] Heuveline, V., Krause, M. J., & Lätt, J. (2009). Towards a hybrid parallelization of lattice Boltzmann methods. Computers & Mathematics with Applications, 58(5), 1071–1080. doi:10.1016/j.camwa.2009.04.001

[8] Melchionna, S., Bernaschi, M., Succi, S., Kaxiras, E., Rybicki, F. J., Mitsouras, D., Coskun, A., et al. (2010). Hydrokinetic approach to large-scale cardiovascular blood flow. Computer Physics Communications, 181(3), 462–472. doi:10.1016/j.cpc.2009.10.017

[9] 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

[10] Wellein, G., Zeiser, T., Hager, G., & Donath, S. (2006). On the single processor performance of simple lattice Boltzmann kernels. Computers and Fluids, 35(8–9), 910–919. doi:10.1016/j.compfluid.2005.02.008

[11] Williams, S., Carter, J., Oliker, L., Shalf, J., & Yelick, K. (2009). Optimization of a lattice Boltzmann computation on state-of-the-art multicore platforms. Journal of Parallel and Distributed Computing, 69(9), 762–777. doi:10.1016/j.jpdc.2009.04.002

[12] CMake – Cross Platform Make. See: http://www.cmake.org/

[13] Visualization Toolkit (VTK). See: http://www.vtk.org/

[14] Virtual Physiological Human Network of Excellence. See: http://www.vph-noe.eu/

[15] Multiscale Applications on European e-Infrastructures. See: http://www.mapper-project.eu/

[16] Hess B, Kutzner C, Van Der Spoel D, Lindahl E (2008). GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. J. Chem. Theory Comput., 2008, 4 (3), pp 435–447

[17] Pronk, S., Larsson, P., Pouya, I., Bowman, G.R., Haque, I.S., Beauchamp, K., Hess, B., Pande, V.S., Kasson, P.M., and Lindahl, E. Copernicus: a new paradigm for parallel adaptive molecular dynamics. In Proceedings of SC. 2011, 60-60.