D2.6.1 – CRESTA benchmark suite

WP2: Underpinning and cross-cutting technologies

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	UEDIN
Version:	0.4
Status	Draft
Author(s):	Jeremy Nowell (UEDIN)
Reviewer(s)	Dan Henningson (KTH), George Mozdzynski (ECMWF)

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	24/01/2012	Outline for comment.	Jeremy Nowell (UEDIN)
0.2	29/02/2012	Draft document for WP2 comments.	Jeremy Nowell (UEDIN)
0.3	02/03/2012	Version for internal review	Jeremy Nowell (UEDIN)
0.4	19/03/2012	Version addressing reviewers comments	Jeremy Nowell (UEDIN)

Table of Contents

Index of Figures

Index of Tables

1 Executive Summary

Benchmarks are widely used in High Performance Computing (HPC). They are used to measure system performance, either to get a general indication of the system's technical capabilities, or to gauge its suitability to a particular application. They are also used to assess the performance of individual applications across a range of systems and HPC architectures. HPC benchmarks therefore cover a wide range of measurements, from low-level computation and communication operations, through computational kernels up to full applications.

In the context of the CRESTA project, benchmarks are useful in two different areas. The first of these is in the understanding the impact of changes to the system-ware on application performance. This requires an assessment of the impact of the changes on the area targeted by them, for example reducing communication latency. However, key to CRESTA are the six co-design applications. Therefore it is also important to understand the impact of system-ware changes on the application performance itself. Secondly, benchmarks are useful to the application developers. They provide a means to assess the impact of changes to the applications, and may also be used to inform design changes in the quest for exascale performance; for instance algorithm choice or the use of different programming models.

The most reliable way of assessing an application's performance is by running the application itself. However, when scaling applications towards exascale it may prove infeasible to do this on a regular basis as changes are made to either system-ware or the application. Therefore an aim of the CRESTA benchmark suite might be to isolate the key computational kernels of the applications so that they may be run on a standalone basis. Unfortunately this has proved infeasible in the initial six months of the project so this remains as a possible goal for the future.

The CRESTA benchmark suite therefore integrates the six CRESTA co-design applications using the Jülich Benchmarking Environment (JuBE), such that they are easy to compile and run. The suite includes different input files suitable for a range of performance measurements.

Along with the applications, the benchmark suite has been configured to run some low-level benchmarks, namely the widely used HPC Challenge (HPCC) benchmarks and the Intel MPI benchmarks (IMB).

The benchmark suite is available to project members from the CRESTA Subversion source control repository, and has been run on one of the main supercomputing platforms of the project, HECToR, which is a Cray XE6 machine.

Future work will focus on working with the co-design application developers to refine the benchmark suite. This will include examining the input files to ensure a representative range of test cases, developing benchmarks of the important computational kernels from the applications, and helping them develop quantitative assessments of the applications performance.

2 Introduction

This document describes the initial release of the CRESTA benchmark suite.

Section 3 describes the motivation for the production of a benchmark suite for CRESTA. Next in Section 4, input to the design of the suite is outlined, taking into account the CRESTA co-design applications. A brief survey of existing benchmark suites is then presented in Section 5. The design and production of the CRESTA benchmark suite is discussed in Sections 6 and 7 respectively. Sample results from the suite are shown in Section 8 before further work is discussed in Section 9.

2.1 Glossary of Acronyms

DEISA	Distributed European Infrastructure for Supercomputing Applications
EC	European Commission
EPSRC	Engineering and Physical Sciences Research Council
FFT	Fast Fourier Transform
HECToR	High-End Computing Terascale Resource
HPC	High Performance Computing
HPCC	HPC Challenge
HPL	High Performance Linpack
JuBE	Jülich Benchmarking Environment
MPI	Message Passing Interface
OpenMP	Open Multi-Processing
PRACE	Partnership for Advanced Computing in Europe
WP	Work Package
XML	Extensible Markup Language

3 Motivation

This section presents the motivation behind the production of a CRESTA benchmark suite. It discusses the purposes of benchmarks both generally, and more specifically from the CRESTA point of view, before describing the different sources of input into the design of the benchmarks.

3.1 What is a Benchmark?

Before discussing the CRESTA benchmarks it is worth considering, briefly, what a benchmark actually is. A benchmark has its origins in surveying, as a permanent mark used to indicate altitude above sea level of that point. The Oxford English Dictionary provides two further definitions [1]:

> *A point of reference; a criterion, touchstone.*
>
> Computing*. A program or set of programs used as a standard against which the performance of other programs (or the computer systems running them) is compared or evaluated.*

Clearly for CRESTA it is the second definition which is relevant, but in which particular context – measuring *program performance* or *system performance*?

3.2 Why Benchmark for CRESTA?

The guiding principle of the CRESTA project is the co-design process. Six key scientific applications have been chosen which have a reasonable expectation of running on future exascale systems. The requirements of these applications are being used to guide development throughout the rest of the project, in a cyclical process. It is therefore very important for the project to understand the impact of any underlying software and hardware changes on application performance. In particular, Task 2.6 of CRESTA is concerned with performance analysis and optimisation, which are key components in the effective utilisation of both current and future HPC systems. It is important to be able to measure the performance of systems so that the impact of changes may be quantified and understood. These measurements may be achieved through the use of benchmarks of *system performance*.

Similarly, it is important to understand how the application design may be modified to make best use of a particular system, for example by different algorithm choice or the use of a different programming model. This may be done by the use of benchmarks designed to measure *program performance*.

The goal of the benchmark suite is therefore to provide a set of measurements useful for both application and system performance, placing particular emphasis on their relevance and usefulness to the co-design applications, and the relationship of the applications with the rest of the project.

3.3 Benchmark Sources

In order to design a CRESTA benchmark suite several sources may be considered to provide input. These are now introduced, before being discussed in detail in the following sections.

3.3.1 Co-design Applications

As described above, the centrality of the co-design applications to the project suggests that the most meaningful results are likely to come from the applications themselves. Unfortunately the complexity of the applications means that it may be infeasible to perform a full run of the application on a regular basis. This is particularly seen to be the case when the fact that the project is concerned with application behaviour at very large-scale is considered. Therefore a large motivation, and indeed challenge, of the benchmark suite is to see if it possible to capture the computational kernels of the applications in such a way that their behaviour is adequately reproduced, whilst at the same time ensuring that they are easy and quick to use for measurements as required by both application and system-ware designers.

3.3.2 Other Applications

In order to guard against any undue bias in the project towards particular features of the CRESTA co-design applications it is important to consider whether any common computational kernels or algorithms are not utilised by them. This may be done by looking at other widely used HPC applications, and input may be sought from other key stakeholders such as the PRACE project, in particular WP7 which is the application focused work package of PRACE.

3.3.3 Low-level Operations

The computational kernels of the applications, both co-design and others, will likely perform particular low-level operations frequently. It is therefore useful to distil these operations from the kernels into individual benchmarks to better inform the project's understanding of both the system and application behaviour with respect to such operations.

4 Input from Co-design Applications

4.1 Application Questionnaire

As described in Section 3.2, the six co-design applications are key to CRESTA, with the benchmark suite aiming to capture their behaviour at large-scale. In order to progress with the benchmark suite the input of the application developers was sought at an early stage. To capture this input a set of questions were put to the developers, with the goal of getting as much information as relevant. The questions are reproduced in the Appendix A.1, and the responses in Appendix A.2. The responses show a good level of engagement, and they are now summarised.

4.2 Questionnaire Results

Study of the questionnaire responses shows that, as already known, the applications are large, complex, scientific codes. Consequently they tend to exhibit a complex behaviour, which often depends implicitly on the problem or data set being analysed.

The full responses of the questionnaire, available in Appendix A.2 are summarised in Table 1.

Table 1: Summary of Application Questionnaire Results

Application	Area	Programming Language	Programming Model	3rd-Party Libraries	Areas of Interest	Existing Benchmark?
ELMFIRE	Plasma Physics	FORTRAN 77/90, C	MPI	BLACS, PESSL, IMSL, (NAG)	Poisson equation	No
GROMACS	Classical Molecular Dynamics	C,C++, x86 assembly, CUDA	MPI, MPI & pthreads	FFTW or similar	Lattice summation, FFTs	No, but used extensively "as is"
HemeLB	Hemo-dynamics using Lattice Boltzmann	C++	MPI (Hybrid MPI & OpenMP in future)	ParMETIS	Comms	No, but want to create one
IFS	Weather Prediction	C, FORTRAN 90/95	Hybrid MPI & OpenMP	LAPACK, BLAS	MPI collectives, global comms, FFTs, Legendre Transforms	Yes - RAPS12
Nek5000	CFD	FORTRAN 77, C	MPI	BLAS	Collectives	No, but some test cases
OpenFOAM	CFD	C++	MPI			No

As the table shows, the key programming languages for application development are C, C++ and FORTRAN, with parallelisation largely provided by MPI or a hybrid model using both MPI and OpenMP. Only IFS have an already existing benchmark suite; all of the other applications obtain performance measurements from running the whole application with different data sets, although HemeLB have a desire to create benchmarks of key computational kernels. Areas of interest cover global communications, particularly MPI collective operations, and Fast Fourier Transforms (FFTs).

4.3 Impact of Questionnaire Results on Benchmark Design

4.3.1 Computational Kernels

Due to the complexity of the applications, and as mentioned by some of the application developers in their responses, it is very difficult to distil their behaviour into a set of key kernels, as desired for a standalone CRESTA benchmark suite. This is particularly the case when considering how much an application's behaviour depends on the problem being analysed. Thus it was decided that the first release of the benchmark suite should concentrate on running the applications themselves, while providing as much performance information as practicable. For this to be interesting to the project the applications must be able to be run with a realistic set of data which illustrates their behaviour at large scale. Where an existing application suite exists, for example RAPS12 for IFS, then this will be used.

4.3.2 Low-Level Operations

Alongside the applications, some low-level benchmarks will be provided for key operations of interest, for example MPI collectives and other global communication patterns. Where appropriate these will be taken from existing benchmark suites.

5 Existing Benchmark Suites

Before implementing the CRESTA benchmark suite a short review of existing HPC benchmarks was performed in order to gain awareness of the current state of the art and avoid unnecessary duplication of existing benchmarks. This review is now presented.

5.1 High Performance Linpack

The High Performance Linpack benchmark [2], or HPL, is used to compile the Top500 list of supercomputers. It solves a dense system of linear equations: $Ax = b$. It relies on the presence of an MPI implementation on the machine being benchmarked. The results give a good indication of the maximum computational performance achievable on a computer; however the computational density is widely accepted as being unrepresentative of applications. It was therefore decided not to include HPL in the CRESTA benchmark suite as a standalone component.

5.2 HPC Challenge Benchmarks

The HPC Challenge (HPCC) benchmark [3] is a suite of benchmarks produced by the same authors as HPL; indeed HPL is one component of the suite. Its goal was to produce benchmarks with more challenging memory access patterns than HPL, providing bounds on real-world application performance by looking at different memory access characteristics. The seven benchmarks included in HPCC are:

- HPL – the Linpack benchmark which measures the floating point rate of execution for solving a linear system of equations, as described above.
- DGEMM – measures the floating point rate of execution of double precision real matrix-matrix multiplication.
- STREAM – a simple synthetic benchmark program that measures sustainable memory bandwidth and the corresponding computation rate for a simple vector kernel.
- PTRANS (parallel matrix transpose) – exercises the communications where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network.
- RandomAccess – measures the rate of integer random updates of memory.
- FFT – measures the floating point rate of the execution of a double precision complex one-dimensional Discrete Fourier Transform (DFT).
- Communication bandwidth and latency – a set of tests to measure latency and bandwidth of a number of simultaneous communication patterns; based on b_eff (effective bandwidth benchmark).

Due to the number of different operations included in the HPCC suite, and its widespread usage in the HPC community, it was decided to include it in the CRESTA benchmarks.

5.3 Intel MPI Benchmarks

The Intel MPI benchmarks [4] provide a comprehensive set of measurements of MPI operations. They cover both MPI1 and MPI2 (including one-sided communication and MPI I/O). With respect to CRESTA they are useful in providing benchmarks of the collective operations, which the applications survey showed to be important, and therefore will be included in the CRESTA suite.

5.4 EPCC OpenMP Microbenchmarks

The EPCC OpenMP benchmarks [5] measure the overheads of OpenMP constructs, including synchronisation, loop scheduling and array operations in the OpenMP runtime library. They compare the parallel execution time of the operation with a serial version as a reference. They are available in both C and FORTRAN. As shown by the application survey no application uses OpenMP as the sole parallelisation method, therefore these benchmarks will not be included in the CRESTA benchmark suite.

5.5 EPCC Hybrid OpenMP/MPI Microbenchmarks

The EPCC OpenMP/MPI microbenchmarks [6] are designed to give low-level performance measurements for a mixed mode OpenMP and MPI programming model. They cover both point-to-point and collective communication patterns. They are available in both C and FORTRAN. The current trend in HPC towards many multi-core nodes, together with the fact that some of the CRESTA co-design applications already use a hybrid OpenMP/MPI programming model means that this suite is more relevant and will be included in the CRESTA benchmarks.

5.6 NAS Parallel Benchmarks

The NAS Parallel Benchmarks [7] are produced by the NASA Advanced Supercomputing Division (NAS). The benchmarks are motivated by Computational Fluid Dynamics (CFD) programs, and are aimed at measuring and evaluating the performance of computer systems rather than the codes themselves. The original benchmarks consisted of several computational kernels and three so-called pseudo applications. These have been enhanced through the addition of benchmarks for unstructured computation, parallel I/O, and data movement. As such they may be considered as something that the CRESTA benchmark suite might work towards, but are unlikely to be useful in themselves as they are not representative of the CRESTA co-design applications.

5.7 Mantevo

Mantevo [8] is a project at Sandia National Laboratories. It aims to provide open-source software packages for the analysis, prediction and improvement of high performance computing applications, and has several goals:

- Predict performance of real applications in new situations.
- Aid computer systems design decisions.
- Foster communication between applications, libraries and computer systems developers.
- Guide application and library developers in algorithm and software design choices for new systems.
- Provide open source software to promote informed algorithm, application and architecture decisions in the HPC community.

It fulfils these goals by writing several so-called “miniapplications” which are designed to be small, self-contained programs that reproduce the behaviour of key scientific applications. By limiting the size of these applications it is possible to experiment with different design choices, library usage and programming models. The miniapplications include:

- HPCCG: Intended to be the "best approximation to an unstructured implicit finite element or finite volume application in 800 semi-colons or fewer."
- pHPCCG: A parametrized version of HPCCG that supports use of different scalar and integer data types, as well as different sparse matrix data structures.
- phdMesh: Parallel heterogeneous dynamic mesh application. Exhibits the performance characteristics of the contact search operations in an explicit finite element application.
- MD: A light-weight molecular dynamics application containing the performance impacting code from LAMMPS.

Further discussion with the CRESTA co-design applications is required to see if any of these might be useful for further study in themselves; however the Mantevo project demonstrates a method of working which the CRESTA benchmark suite should aim towards; developing small, standalone kernels.

6 CRESTA Benchmark Suite Design

The benchmark suite will initially contain the six CRESTA co-design applications, along with relevant low-level benchmarks. The simplest solution would be to package these together with instructions for running them. However it was decided that a more useful approach would be to try and use some kind of framework as a wrapper. The natural candidate for this would appear to be JuBE, the Jülich Benchmarking Environment [9], from the Jülich Supercomputing Centre of Forschungszentrum Jülich (FZJ), Germany. JuBE was used for both the DEISA Benchmark Suite [10] and the PRACE Benchmarks [11],[12]; therefore there is existing expertise on its usage amongst some of the project partners.

6.1 JuBE: Jülich Benchmarking Environment

The JuBE benchmarking environment provides a script-based framework for creating benchmark suites, utilising Perl and XML. Once configured it allows the compilation, execution and analysis of benchmark results. It may be easily configured for multiple platforms. The architecture is shown in Figure 1. Note that the analysis GUI has not yet been implemented.

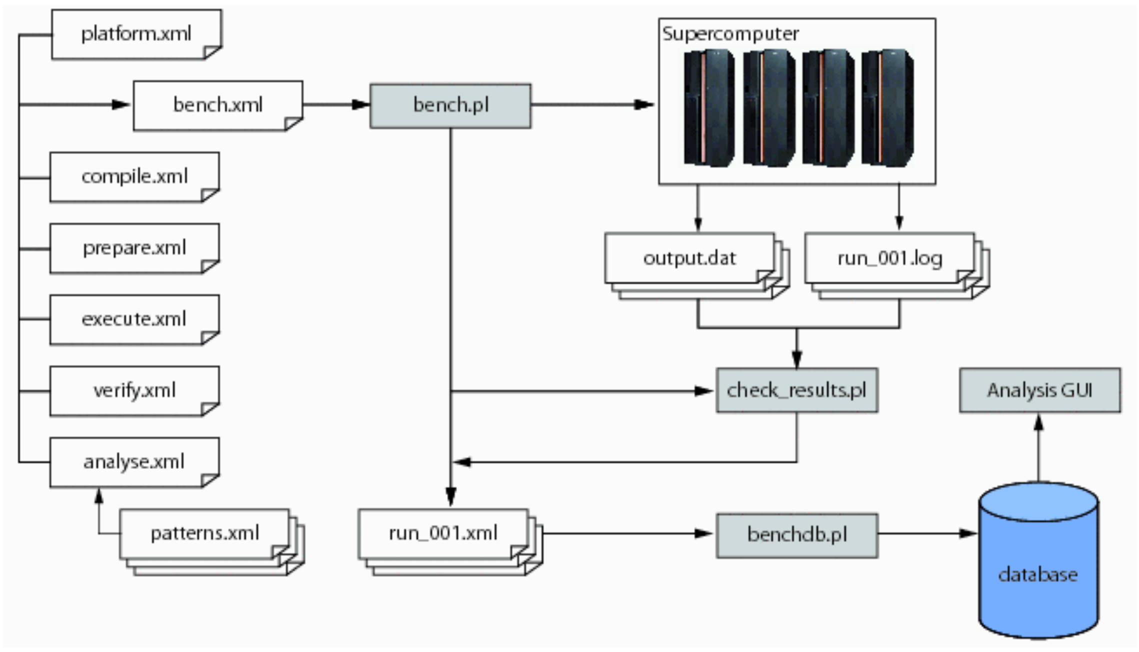

Figure 1: Architecture of JuBE.

Each benchmark application is configured using several XML files:

- bench.xml – the top-level configuration that contains details of the benchmark runs for and individual benchmark and platform combination.
- platform.xml – contains details of each compute platform the benchmark has been configured for. This includes details such as compiler settings, batch system details and third-party library locations.
- compile.xml – how the benchmark should be compiled.
- prepare.xml – how to setup the benchmark input files for each benchmark run. This is typically done by substitution of parameters into a template file.
- execute.xml – how to run the benchmark, typically by submission to a batch system. In this case it uses a template of a batch script for the platform.
- verify.xml – how to verify that a particular benchmark has been successfully executed.
- analyse.xml – how to obtain measurement data from the benchmark output. This utilises the patterns.xml file.

7 Production of Benchmark Suite

This section describes the production of the benchmark suite, particularly its component parts such as the co-design applications. It briefly describes the work necessary to port each application to one of the main computational platforms of the project, EPCC's Cray XE6, HECToR [13]. Although the benchmark suite has not yet been tested on any of the other CRESTA platforms this will be done shortly. It should not be too much work as the main CRESTA machines are very similar Cray variants.

7.1 Application Benchmarks

In this first release of the benchmark suite the application benchmarks consist of the JuBE configuration files for each of the six CRESTA co-design applications, along with input files required to run different simulations. The input files should cover a range of cases; a verification case, a realistic case, and a challenging case. The realistic case and challenging case should be able to be used for measuring the current performance of the application, and therefore provide a means of assessing the applications scalability.

When measuring scalability two different methods can be used, strong scaling and weak scaling. In strong scaling measurements the problem size is kept fixed but the number of processors is increased. Good strong scaling is often seen as the hardest to achieve, as usually the communication overhead increases as the number of processors. In weak scaling measurements the amount of work assigned to each processor stays constant and the problem size is increased. Within CRESTA the emphasis will be on increasing the strong scaling of the co-design applications, therefore this should be measurable using the CRESTA benchmarks.

7.1.1 ELMFIRE

ELMFIRE is a first principle plasma turbulence simulation code with full-function gyrokinetics. At present the ELMFIRE code achieves scalability of up to 2,048 processors.

7.1.1.1 Porting Details and Changes Required

ELMFIRE had already been ported to CSC's Cray machine Louhi [14], a Cray XT4/5. Thus, it was very simple to port to HECToR, with the work being performed by the developers. All that was required was a new Makefile for HECToR, which is in fact identical to the Louhi Makefile. This gives a good level of confidence that the code will run on the other CRESTA platforms.

7.1.1.2 Benchmark Description

Several input files were provided by the ELMFIRE developers for running benchmarks, representing a typical simulation that is run currently. The simulations represent a torus/tokamak that has a major radius of 55cm and a minor radius of 8cm. The largest version of this simulation uses a grid of size 120x150x8 to simulate a plasma containing approximately 470 million ions and electrons for a time period of 300ns in 30ns steps. (Real simulations would run for many more steps). Other input files are also available with scaled down versions of the simulation.

The ELMFIRE output provides timings for the simulation, broken down by the various stages.

7.1.2 GROMACS

GROMACS is a major open source software package for biomolecular simulation, developed by an international collaboration steered from KTH, Sweden.

7.1.2.1 Porting Details and Changes Required

No changes were required to port the code to HECToR.

7.1.2.2 Benchmark Description

The GROMACS developers provided three different benchmarks, each representing a different system. These are now described.

7.1.2.2.1 Ion Channel System

The ion channel system is the membrane protein GluCl, which is a pentameric chloride channel embedded in a lipid bilayer. This system contains roughly 150,000 atoms, and represents a challenging parallelisation case due to its small size, and useful for measuring the strong scaling of GROMACS. However, it is one of the most desired target sizes for biomolecular simulations due to the importance of these proteins in pharmaceutical applications.

It is particularly challenging due to the highly inhomogeneous and anisotropic environment in the membrane, which poses hard challenges for load balancing by domain decomposition.

The baseline simulation is performed with standard PME lattice summation for electrostatics, and cut-offs at 1.0 nm.

The default simulation uses 2.5fs time steps and constrained bonds; there is also a simulation with vsites and 5fs time steps.

7.1.2.2.2 Methanol

This benchmark is a system of 1.28 million methanol molecules in a cubic box, 7.7M atoms, using Particle Mesh Ewald (PME) calculations. It is useful for assessing the weak scaling of GROMACS.

This system is intended to run at a PP(Particle-Particle):PME process ratio of 3:1; this has been tested with Gromacs version 4.5 on x86 systems, on other systems or with newer Gromacs versions this might differ. At the end of the md.log file the balance will be shown and a note is printed when the PP/PME imbalance is large.

The PP/PME balance can be tuned by scaling rlist,rcoulomb,rvdw and fourier_spacing in grompp.mdp by the same amount.

7.1.2.2.3 Vesicles

This benchmark consists of a two lipid vesicles tethered by a small chemical linker, and is used to study fusion. The system size is quite large, roughly 2 million atoms, and it is very slow to simulate on normal workstations.

It uses a 4fs time step, virtual sites, and PME lattice summation.

This system, like the ion channel system, is highly inhomogeneous and anisotropic, but due to its size it will scale better. It has been run successfully over thousands of cores on the Cray XE6 at KTH.

Since it contains a lot of water, it is a relatively tough test for load balancing in particular (the vesicle part is slower to calculate).

7.1.3 HemeLB

HemeLB is a high performance lattice Boltzmann code for simulating fluid flow in sparse geometries, such as those found in the vasculature.

7.1.3.1 Porting Details and Changes Required

Due to licensing issues HemeLB has not, at the time of writing, been integrated into the CRESTA benchmark suite, however it is hoped to rectify this situation shortly. The code has been ported to HECToR successfully and run by the HemeLB developers.

7.1.3.2 Benchmark Description

A relatively simple benchmark simulating flow within a cylinder has been setup and run. A benchmark for a sparse geometry will also be setup.

7.1.4 IFS
The Integrated Forecast System (IFS) is an integrated set of applications written by ECMWF for Numerical Weather Prediction. RAPS12 is IFS provided as a benchmark suite for measuring its performance on different systems.

7.1.4.1 Porting Details and Changes Required
Minor changes were required to get the code built and running on HECToR using the Cray compilers. These may be seen in detail in the benchmark configuration files. RAPS12 is licensed code, therefore is not available with the benchmark suite, but must be obtained separately from ECMWF. The suite contains the configuration files required for compiling and running it once downloaded.

7.1.4.2 Benchmark Description
RAPS12 is provided with several model datasets at different resolutions. The simplest of these may be used for verification tests, whilst the others are more challenging. RAPS12 provides extensive profiling information using its own timer package as part of each run.

7.1.5 Nek5000
Nek5000 is a highly scalable open-source simulation code for the simulation of incompressible flows in moderately complex geometries. The discretisation is based on the spectral-element method (SEM), which combines the high-order accuracy of spectral methods with the geometric flexibility of finite-element methods.

7.1.5.1 Porting Details and Changes Required
No changes were required for compiling Nek5000 on HECToR.

7.1.5.2 Benchmark Description
Since the spectral element method is employed in the Nek5000, the execution time of Nek5000 mainly depends on tensor products for multi-dimensional problems. Nek5000 is implemented using matrix-matrix products for the tensor products required for the high-order spectral methods. As a result the operations reduce from $O(N^{2d})$ to $O(N^{d+1})$ for d-dimensional problems.

One of the examples included in Nek5000 is a timing benchmark for exploring the efficiency of the matrix-matrix products on a particular system.

7.1.6 OpenFOAM
OpenFOAM is an open-source package for computational fluid dynamics using the Finite Volume Method. It comes with a wide-range of solvers for different physical systems.

7.1.6.1 Porting Details and Changes Required
Minor changes were required to compile OpenFOAM on HECToR. These were necessary to use the correct compilers and settings.

7.1.6.2 Benchmark Description
The benchmarks chosen for OpenFOAM were taken from the examples included with OpenFOAM. For small test cases the commonly used lid-driven cavity flow and dam break examples were used. After discussions with local experts it was decided to use 3D versions of these examples as developed and described in [15]. For the challenging case it was decided to use a simulation of flow round a motorbike, using a mesh containing 32 million cells.

7.2 Low-Level Benchmarks

For the moment the HPC Challenge and the Intel MPI Benchmarks are included in the CRESTA benchmarks, as these were the easiest to integrate. No modifications were made to either package; they were included as is.

7.3 Benchmark Instructions

An overview of running a benchmark is described here. For full details see the README file included in the benchmark description.

First, obtain the benchmark suite from the CRESTA SVN repository[1], available at the following location:

```
https://svn.ecdf.ed.ac.uk/repo/ph/cresta/wp2/benchmarks
```

Note that if either of the licensed codes, HemeLB or IFS RAPS12, is required then these must be obtained separately and installed in the `src` directory of the benchmark suite.

Next it is simply a case of compiling and running the benchmarks by means of simple perl commands. For example to run OpenFOAM:

```
bash-2.05a$ cd DEISA_BENCH/applications/OpenFOAM

bash-2.05a$ perl ../../bench/jube bench-Cray-XE6-HECToR.xml
```

Once a run is complete the results may be verified and analysed as follows:

```
bash-2.05a$ cd DEISA_BENCH/applications/OpenFOAM

bash-2.05a$ perl ../../bench/jube -update -result <ID>
```

Where ID is the reference number assigned by JuBE to a particular run.

[1] Note that this is only open to registered CRESTA members. Contact the author for access to the SVN.

8 Sample Benchmark Results

This section contains some sample benchmark results obtained on HECToR. Note that this is not intended to be a comprehensive collection of results, rather an indication of what is achievable with the benchmark suite.

Figure 2 shows the results obtained with the HemeLB benchmark of a simple cylinder, containing over 15 million lattice sites, for different numbers of cores. The results are measured in terms of a velocity, which is defined as site updates per second per core. Perfect strong scalability would be indicated by a flat line. This demonstrates how the benchmark suite may be used for measuring the applications scaling.

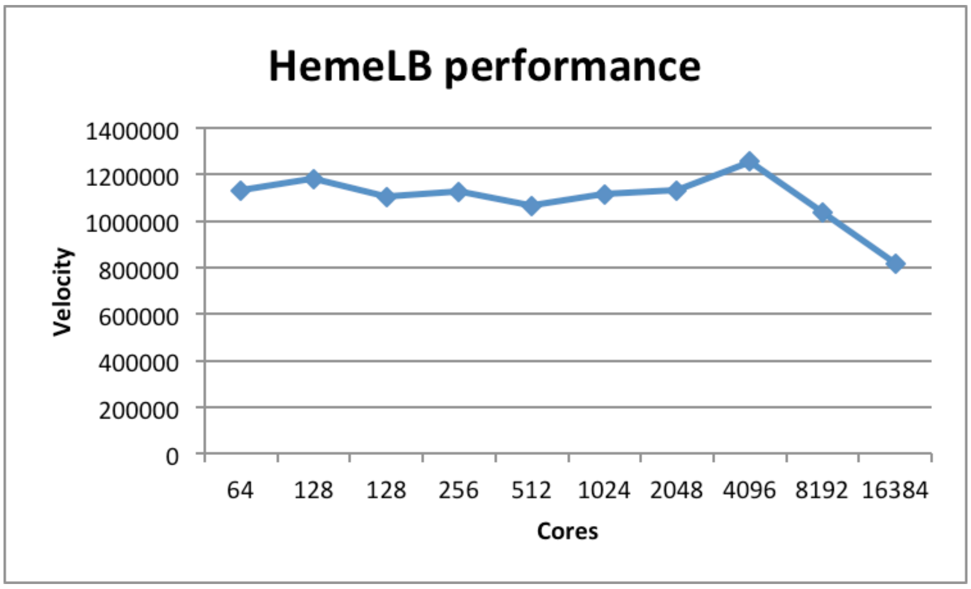

Figure 2: Performance of HemeLB benchmark on HECToR. Velocity is defined as site updates per core per second.

9 Further Work

There are several options for further work on the CRESTA benchmark suite. Clearly these must be decided in collaboration with the rest of the project, in order to be as useful as possible.

The suite will be kept up to date with the latest versions of the applications as decided by the application developers within WP6. Also the benchmark files themselves will be regularly assessed to ensure they are still relevant. This will be particularly the case for the “challenging” benchmarks to make sure that the limits of scalability of the applications are being explored properly. The benchmark suite can be used for scaling and efficiency studies of the applications. They provide an opportunity to develop a set of criteria for assessing the development of the application performance, and these will be developed in collaboration with WP6 if felt to be appropriate.

The suite will be ported to different architectures available for use within the project. As already discussed the main platforms are all Cray machines, however if possible access to other machines will be obtained in order to provide information on application performance across architectures.

The suite will be extended with the addition of new benchmarks. These could cover particular algorithms under consideration by the applications or being studied by the system-ware developers. As discussed in D3.1 [16], it is likely that application developers will want to study different programming models and parallelisation techniques, therefore the benchmark suite could be extended to study particular algorithms or operations in different programming models.

The feasibility of extracting the computational kernels from the applications will be pursued. The report [17] from WP5 of EPSRC’s Architecture Comparison Exercise (ACE) [18] illustrates some of the difficulties associated with trying to relate application performance to algorithm benchmarks. This area of work will require further discussions with WP6 to decide on the best way forward.

10 References

[1] Oxford English Dictionary. Second edition, 1989; online version September 2011. http://www.oed.com/view/Entry/17612; accessed 12 December 2011.

[2] High Performance Linpack, http://www.netlib.org/benchmark/hpl/

[3] HPC Challenge Benchmark, http://icl.cs.utk.edu/hpcc/.

[4] Intel MPI Benchmarks, http://software.intel.com/en-us/articles/intel-mpi-benchmarks/.

[5] EPCC OpenMP Microbenchmarks, http://www2.epcc.ed.ac.uk/computing/research_activities/openmpbench/openmp_index.html

[6] EPCC OpenMP/MPI Microbenchmarks, http://www.epcc.ed.ac.uk/software-products/openmpmpi-microbenchmarks

[7] NAS Parallel Benchmarks, http://www.nas.nasa.gov/publications/npb.html

[8] Mantevo, https://software.sandia.gov/mantevo/

[9] JuBE: Jülich Benchmarking Environment, http://www2.fz-juelich.de/jsc/jube/

[10] "Initial Report on the DEISA Benchmark Suite", DEISA2 Deliverable DEISA2-D7-2.1, http://www.deisa.eu/publications/fp7-deliverables/PM06/DEISA2-D7-2.1.pdf

[11] "Report on available Performance Analysis and Benchmark Tools, Representative Benchmark", PRACE Deliverable D6.3.1, http://www.prace-project.eu/IMG/pdf/D6-3-1.pdf

[12] "Final Benchmark Suite", PRACE Deliverable D6.3.2, http://www.prace-project.eu/IMG/pdf/D6-3-2-extended.pdf

[13] HECToR, http://www.hector.ac.uk/

[14] Louhi, http://www.csc.fi/english/research/Computing_services/computing/servers/louhi

[15] Gavin J. Pringle, "Porting OpenFOAM to HECToR – A dCSE Project", HECToR CSE Report, http://www.hector.ac.uk/cse/distributedcse/reports/openfoam/

[16] "State of the art and gap analysis", CRESTA Deliverable D3.1

[17] Architecture Comparison Exercise 2, WP5, "An HPC Roadmap: Understanding the Performance of Applications across a Range of Architectures", http://www.epsrc.ac.uk/SiteCollectionDocuments/other/ACE2WP5HPCRoadmap.pdf

[18] Architecture Comparison Exercise, http://www.epsrc.ac.uk/ourportfolio/themes/researchinfrastructure/subthemes/einfrastructure/hpc/Strategy/Pages/ace.aspx

Annex A. Application Questionnaire

A.1 Questions

This section details the questions that were put to application developers in order to get their input to the benchmark suite.

Q1. Your name?

Q2. Your application?

Q3. What are the key algorithms which determine the scaling of your application? Please provide as much detail as possible, for example:

- the programming model,
- the language,
- the impact on scaling of the algorithm,
- the effect of different problem sizes,
- the communication patterns utilised.

Q4. Please indicate which, if any, third-party libraries you utilise in these algorithms.

Q5a. Do you already have a benchmark framework which tests these algorithms? Please explain if this benchmark runs the complete application or whether it is possible to just run these key computational algorithms.

Q5b. If so, would you be able to provide us with the benchmarks? Are there any licence restrictions on the re-use of this code?

Q6. Do you use, or know of, any third-party benchmarks which provide a good indication of your application's performance?

Q7. If you do not have an existing benchmark, would you be able to provide us with the computational kernel of your code and instructions on how it is used, such that we could turn this into a representative benchmark?

Q8. Looking towards the future, do you see the algorithms your application uses changing significantly? Which other algorithms would you like to see included in the benchmark suite in order to help your design process?

Q9. Do you have any other comments or information that may be useful in us producing the benchmark suite?

A.2 Responses

This section details the response from the application developers, the answers numbered according to the questions listed in Section A.1.

A.2.1 Elmfire

A2. Elmfire, simulating the time propagation of extended charged particles (a few grid units) in the background of a magnetic and an electric field. The magnetic field is static and the electric field is periodically updated by solving Poisson's equation for the electrostatic potential on a grid from the positions of the charged particles.

Åbo is not the code owner of Elmfire. We are presently working with the code owners VTT and Aalto University to create a document that describes the computational aspects of Elmfire from a modern parallel programming point of view. This will enable us to focus on algorithmic choices rather than physics requirements.

A3. The particles are owned by the processes, likewise the electric potential grid. Therefore there is an all-to-all communication before solving the Poisson equation where the source term is gathered. Likewise, the solution of Poisson's equation is done collectively and iteratively. We believe Elmfire today is bound by its communication patterns.

A4. PETSc. Elmfire can also use PESSL as it was originally built on an IBM supercomputer.

A5a. The formulation and the solving of the Poisson equation is tightly integrated into the code today. Possibly this part could be extracted and used as a benchmark program?

A5b. There are no particular licence restrictions except that Elmfire may be used only within the CRESTA framework and duration.

A8. We would be happy to redesign central parts of Elmfire, e.g. towards a complete domain decomposition with the passing of particles between processes.

A.2.2 Gromacs

A2. Gromacs. Classical molecular dynamics.

A3.

- *the programming model,*

The computational bottleneck is the calculation of fairly simple interactions between particles in spatial proximity, but since these particles move in space we need fairly advanced logic to track them. This either ends up as a list-of-neighbours (verlet neighbourlist) or more advanced algorithms where we reduce the algorithm to interactions in "subtiles" where all-vs-all particles interact (which provides better memory access patterns).

Historically we have used a pure MPI model. Over the last year we have experimented with mixed MPI-openMP, but based on this we are likely headed to mixing MPI with pure pthreads. OpenMP is hard to control on a level low enough.

- *the language,*

C/C++/x86 SIMD assembly intrinsics/Cuda

- *the impact on scaling of the algorithm,*

Weak scaling in Gromacs is perfect for the basic algorithm when using simple cutoffs, but we have some remaining bottlenecks for handling e.g. initial I/O for the entire system. Our real scaling challenges today have to do with electrostatics where we need lattice summation algorithms and FFTs, and strong scaling.

- *the effect of different problem sizes,*

The limit we care most about is that we can go down to roughly 250 atoms per core today, which is the bottleneck for our strong scaling. We would like to push this further.

- *the communication patterns utilised.*

Pulsed or asynchronous communications with next neighbours. We use "neutral territory" domain decomposition that is pretty much the state-of-the-art today.

Lattice summation is run on a separate set of nodes (roughly 20%) to improve FFT parallelization. This involves some all-to-all communication over subcommunicators by definition (we use a 2D pencil decomposition of the FFT grid), and one area we are exploring is multigrid solvers to avoid this.

A4. FFTW or some other FFT library, but the actual FFT isn't the bottleneck for the parallel version; the FFTs are typically quite short, so it's more the communication patterns limiting us.

A5a. We have a bunch of test systems we can share, and we can easily create larger ones. It is by far the easiest to run the entire application.

A5b. Yes. No restrictions whatsoever (LGPL).

A6. Gromacs is a pretty widely used application, and there are several people that have done benchmarks, see e.g.

http://www.cse.scitech.ac.uk/cbg/benchmarks/Report_II.pdf

In general, as you can see in these benchmarks, both the blessing and curse or Gromacs is that we achieve very high performance on a given number of nodes, which then by definition (Amdahl's law) hurts our relative scaling. Still, for applications all the users care about is of course absolute performance!

A7. See above. One caveat is that we want to optimize for real simulations, and then it is important to include effects of load balancing and separate lattice summation code paths. It's better to use the full application to reflect this.

A8. Yes. We are currently getting rid of the verlet-style neighborlists and introducing more streaming-friendly kernels. This is particularly important for GPUs, and we see combined CPU-GPU parallelization as critical for the future, including the new Cray XK6 nodes.

A9. Decide now whether you want a benchmark suite that primarily tries to make the project look good, or whether we also need some tougher internal benchmarks that deliberately aim to expose the bottlenecks to increase absolute performance (which might lead to different choices compared to a goal of just increasing relative scaling).

A.2.3 HemeLB

A2. HemeLB (Hemodynamics with Lattice Boltzmann)

A3. HemeLB is written in C++ and currently is flat MPI, but we will be implementing a hybrid MPI/OpenMP version in the early part of CRESTA. It contains several parts: 1, the core LBM; 2, the renderer; 3, the steering; 4, IO; and 5, inlet/outlet boundary conditions.

1) As it is a lattice-Boltzmann code; this implies several things:

- the simulation domain is made up of regularly spaced sites
- the state of a site at time $t + \Delta t$ is a function only of that site and its neighbours at time t

The communication patterns are therefore between neighbouring domains only. HemeLB however does not operate on a simple rectangular domain: since for blood flow applications only a small fraction (5–10%) of the space spanned by the vessels is typically occupied by fluid, we only create sites within the fluid. This is therefore not a simple Cartesian pattern as often used by LB implementations.

2) The renderer has a more complex communication pattern. At the moment we use a binary tree pattern of non-blocking send/recv calls to composite the final image over a number of timesteps O(log(p)). This leverages a coalesced communication system we have implemented within the code.

3) The steering component effectively has to broadcast its data to all MPI tasks. To avoid the poor scaling of collectives, we use the same machinery used by the renderer but in reverse to pass the steered parameters to the tasks that require it.

4) For writing snapshots, we use MPIO. Each MPI task writes a record for each fluid site it holds to a buffer which is then sent to file through MPIO in a fairly simple way (MPI_File_write_all)

5) The inlet/outlet BCs are sent asynchronously from the master task to all the workers that need an individual in/outlet's data.

We note that we hope to refine points 2 and 3 through our work in WP5.

A4. We use ParMETIS during the initial domain decomposition only.

A5a. We currently do not have robust benchmarks, but are actively engaged with development of whole-application performance measurement benchmarks. We hope to develop key-algorithm benchmarks ourselves in the near future, covering both the scientific kernel and the visualisation and steering components, which are also expensive in terms of both communication and computation.

A6. No

A7. As stated in Q5a above, we are currently engaged with development of benchmarks ourselves. We do not wish to pass this objective over in its entirety to CRESTA collaborators, but would expect to be involved in integration of our work into a CRESTA benchmark suite as part of our contribution to CRESTA as a co-design application.

A8. We do not see the algorithms themselves changing, but the implementations will change at least somewhat during the hybridisation process.

One of the largest limitations on LB performance is currently the memory bandwidth between cores and RAM. We would like to see benchmarks that stress this component of architectures and coping technologies such as prefetching etc. Depending on implementation, LB codes either read a contiguous part of an array and write to a non-contiguous, scattered part or vice-versa; we would like to see benchmarks that have comparable characteristics.

Q9. As stated above, we would expect to be actively involved in integration of our benchmarks into the suite, at both the development and review levels. We plan for our whole-application level performance measurement capabilities to be complete in the first quarter of 2011, a time scale that should be compatible with the CRESTA deliverable. James Hetherington would hope, at the beginning of 2012, to identify and meet a developer within the benchmark team, and work closely with this individual to develop this part of the benchmark.

A.2.4 IFS

A2. Integrated Forecasting System (IFS).

The IFS is a Spectral Numerical Weather Prediction (NWP) model. It has over 1 million lines of code.

A3. A spectral model by definition involves global communication, using MPI collective calls where possible, such as MPI_allgatherv.

At the present operational resolution of IFS (T1279), MPI communications account for about 10 percent of the total wall clock time.

Some key algorithms used in IFS are:

- Legendre Transform (LT)
- Fast Fourier Transform (FFT)
- Semi-Lagrangian Scheme (SL)

- *the programming model,*

Hybrid MPI/OpenMP

- *the language,*

Fortran 90/95, C

- *the impact on scaling of the algorithm,*

LT: $\mathcal{O}(N^3)$ cost, uses DGEMM

FFT: $\mathcal{O}(N^2 \log N)$ cost

SL: non-local MPI communication

where N is the spectral truncation

- the *effect of different problem sizes,*

Halving the grid spacing in the model, typically results in about a 16 times increase in model cost (3 spatial dimensions + time step)

- *the communication patterns utilised.*

Global (per MPI communicator group), and local communications

A4. LAPACK and BLAS

A5a. We have a benchmark framework which runs the complete application with an extensive set (some 2000) of timers which are grouped into:

- OpenMP parallel regions
- MPI communication
- Serial (i.e. non-OpenMP)

Timers are summarised at the end of a run per MPI task and over all tasks.

A5b. The benchmark code (the latest version is called RAPS12) is available to CRESTA partners, although a licence agreement is required to be signed.

A6. No.

A7. Not applicable.

A8. Not at this point in time.

A9. ECMWF are currently porting the latest IFS model benchmark called RAPS12 to HECTOR, with resolutions up to T2047 which is expected to be the next ECMWF operational resolution in 2015. A T3999 sized model case is also being prepared, although for practical reasons, this would necessitate the Fast Legendre Transform development to be available in 2012.

A.2.5 Nek5000

A2. NEK5000 is computational fluid dynamics solver based on the spectral element method (SEM). This numerical scheme combines the geometric flexibility of finite elements with the minimal numerical dispersion and dissipation of spectral methods. In the SEM, the solution within each of the individual elements is represented as a tensor-product of Nth-order Lagrange polynomials based on the Gauss-Lobatto-Legendre nodal points. Typical applications for NEK5000 include the large-scale parallel simulation of turbulent flows in moderately complex geometries.

Summary: http://www.mcs.anl.gov/~fischer/nek5000/fischer_nek5000_dec2010.pdf

A3.

- the programming model, the language,

The code is written in Fortran77/C and employs the MPI standard for parallelism. So far no OpenMP/threading implemented. Array allocation is static (compile-time).

- the impact on scaling of the algorithm,

The principal computational bottleneck arises in simulating unsteady incompressible and low-Mach number flows. In these cases, the elliptic problem governs the pressure, which must be computed implicitly at each time-step. For large unstructured problems, the resultant discrete Poisson problem is most efficiently solved using multilevel iterative methods embedded within a Krylov subspace projection scheme such as conjugate gradients or GMRES. The multigrid/GMRES/CG solvers require the use of parallel collective operations limiting the scalability of the code when an elliptic solver is in use.

- the effect of different problem sizes

NEK 5000 showed excellent scaling up to 280,000 processes (http://nek5000.mcs.anl.gov/index.php/Scaling). If a linear solver is in use (elliptic problem), then the code scaling is affected by the scaling of the collective operations ($\log P$, where P is the number of processes).

A3. MPI, BLAS.

A5a. Nek5000 is distributed with automated build/test suite and with an example suite. Such input-files can be used as test cases for a benchmark activity. However, for CRESTA we suggest a larger problem (turbulent pipe flow) as the main scaling example, and the jet in crossflow as the benchmark for adaptivity.

A5b. The benchmarks are included in the Nek5000 distribution. The additional suggested cases will be delivered separately.

Nek5000 is an open-source code, released under GPL.

A6. Nek5000 has been included in a IBM Bluegene benchmark some years ago.

A7. Yes.

A8. In the CRESTA project, three different strategies will be studied:

i) we intend to extend Nek5000 to support p-type (i.e., variable approximation order) adaptivity. The spectral element method used in Nek5000 readily admits variation in approximation order since all operators are evaluated in matrix-free form and the order can be changed with no pre-processing overhead. The proposed p-refinement strategy will have no impact on the mesh topology, which implies significant simplifications concerning the Nek5000 multilevel preconditioners.

ii) alternative discretisation in one or two directions based on modal decomposition using Fourier series.

iii) In particular for the Fourier decomposition, a hybrid parallelisation using OpenMP and MPI might be beneficial. So far, no threading is being used in Nek5000.