PRACE Application Enabling Work at EPCC

Xu Guo, *EPCC, The University of Edinburgh*
and
Joachim Hein, *EPCC, The University of Edinburgh*

ABSTRACT: *The Partnership for Advanced Computing in Europe (PRACE) created the prerequisites for a pan-European HPC service, consisting of several tier-0 centres. In 2010, the PRACE project will move to the Implementation Phase. The now completed work prepared all the necessary legal, administrative, and technical work for the pan-European service implementation. This paper discusses the software enabling work done in PRACE Work Package 6 by EPCC on the PABS application codes NAMD and HELIUM, with a particular focus on the work carried out for the Cray XT5 prototype system during the PRACE Preparatory Phase. This paper also includes a performance comparison with non-Cray systems available to PRACE.*

KEYWORDS: PRACE, application enabling, NAMD, HELIUM, Cray XT5, EPCC

1. Introduction

PRACE

PRACE, The Partnership for Advanced Computing in Europe, aims to provide the European researches with a persistent pan-European HPC service to enable world-class science [1]. The Partnership, originally consisting of 16 partners from 14 countries, now has total 20 members involved [2]. The PRACE project Preparatory Phase was part-funded by the EU's 7th Framework Programme (FP7/2007-2013) under grant agreement n° RI-211528, which started from January 2008 with a duration of 2 years [3]. During the Preparatory Phase, PRACE looked into all aspects of the contractual and organisational issues, the system management, application enabling and future computer technologies for the pan-European HPC service. The PRACE Implementation Phase will start in June, 2010.

PRACE WP6 and PABS

The Work Package 6 (WP6) of the PRACE Preparatory Phase was responsible for the applications enabling on future petascale systems. Its primary goal was to identify and understand the software libraries, tools, benchmarks and skills required by users to ensure that their application can use a Petaflop/s system productively and efficiently [4]. WP6 was the largest technical activity in the PRACE Preparatory Phase, which most of the partners involved in. EPCC carried the overall responsibility for WP6 and was heavily involved in the technical work.

WP6 selected a set of representative applications from a wide range of applications into the PRACE Application Benchmark Suite (PABS) [5]. The PABS was created based on the actual European HPC usage, the coverage of scientific areas, the applications scalability and the performance on different architectures. The final PABS consists of 22 applications in total, including ALYA, AVBP, BSIT, Code_Saturne, CPMD, CP2K, ELMER, GADGET, GPAW, GROMACS, HELIUM, NAMD, NEMO, NS3D, OCTOPUS, PEPC, QCD, Quantum_Espresso, SPECFEM3D, TORB/EUTERPE, TRIPOLI-4, and WRF [5].

During the Preparatory Phase, these applications were ported to a number of PRACE prototypes and their requirements for petascale architectures were captured. The applications' scalability and possible optimisations for petascaling were also investigated in the WP6 activities. EPCC was mainly responsible for the application work on NAMD and HELIUM, with

significant contributions from other PRACE partners, relating to the non-Cray prototypes.

PRACE Cray XT5 prototype

The PRACE prototypes are a set of systems which were selected to represent the current and emerging technologies most suitable for supercomputer architectures. The selected current systems comprise MPP systems, thin- and fat-node clusters, vector systems and system using the Cell processors. This paper focuses on the study of NAMD and HELIUM on the Cray XT prototype Louhi, which is located at CSC, Finland.

Louhi is one of the most widely used MPP prototypes for the investigations of PRACE WP6. The system offers Cray XT4 and XT5 nodes. The XT5 portion of the system has 672 nodes each containing two quad-core 2.3 GHz AMD Opteron 64-bit Barcelona processors, and 180 XT5 nodes (belonging to the PRACE project) each containing two quad-core 2.7 GHz AMD Opteron 64-bit Shanghai processors. The memory size per core is 1GB or 2GB. Its communications network utilises the Cray SeaStar2 communication system. The nodes are arranged in a 3D torus [6].

About this paper

This paper looks into the software enabling work done by EPCC for the application NAMD and HELIUM on the PRACE Cray XT5 prototype, Louhi. Both of the codes were ported successfully to Louhi during the Preparatory Phase. The porting experiences, scalability and performance are described in this paper. The optimisation strategies for petascaling were investigated for the two codes and the effects are discussed here. Besides Louhi, the performance of NAMD and HELIUM is also compared on a number of other PRACE prototypes, including Huygens, an IBM Power6 system at SARA, Netherlands; JUGENE, an IBM BlueGene/P system at FZJ-JSC, Germany; and JuRoPa, a Sun x86 cluster with the Nehalem processors, located at FZJ-JSC, Germany.

The Section 2 of this paper mainly discusses on the work done for the application NAMD and Section 3 is for the application HELIUM. A conclusion is given in section 4.

2. NAMD

Overview

NAMD is a molecular dynamics application designed to simulate bio-molecular systems [7, 8]. It is widely used and can be deployed on a wide variety of compute platforms. NAMD is developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. The application design places a strong emphasis on scalability, allowing a large number of processors to be deployed efficiently for a single calculation. The application is quite flexible with respect to the data format of the input files. It can read a wide variety of the commonly used file formats for e.g. force fields, protein structure etc.

NAMD 2.7β1, released in March 2009, was used for the study in the PRACE Preparatory Phase. The second beta version 2.7β2 was not released until November 2009, towards the end of the PRACE Preparation Phase. It was decided to keep investigating NAMD 2.7β1 to keep everything consistent.

The NAMD source is written in C++ using Charm++ parallel objects [9] for the data exchange between the compute tasks. Building Charm++ is typically the first step when building a NAMD executable. The source for Charm++ 6.1 is included in the NAMD 2.7β1 source. In addition a production version of NAMD requires a TCL and a single precision version of FFTW 2.1.5.

For the force calculation NAMD uses a cut-off distance. The cut-off is specified in the input files. For atoms separated by less than the 0cut-off the forces are calculated directly in position space. If atoms are separated by more than the cut-off, only the long range electro-static forces are considered. These are calculated using the particle mesh Ewald (PME) method.

NAMD is parallelised using a spatial decomposition. The simulation volume is divided into orthorhombic regions called patches [7]. The diameter of these patches has to be larger than the cut-off distance. Hence for the calculation of the direct forces, knowledge of atom position on the home patch and the 26 neighbouring patches is all that is required.

NAMD automatically adjusts the load balance during the first part of the simulation. The computational load is measured for each patch and patches are moved between the processors to balance the load. Most of the code required for the load balancing features is part of Charm++ instead of the actual NAMD source. The load balancing takes the first 300 time steps of a simulation. These slower initial steps need to be taken into consideration when estimating the performance of a large production run from a test simulation lasting only a few hundred steps.

Building and Running NAMD on the Cray XT

To build NAMD on the Cray XT system, the gcc 4.3.3 compiler was used. The Cray XT is very well supported by the Charm++ and NAMD development team. To build Charm++ 6.1, the description file `mpi-crayxt` was used, which is part of the Charm++ 6.1 distribution.

The "standard" Cray software stack does not provide a TCL library, so it requires building as well. There was no issue when building TCL version 8.4.19 using the gnu compiler. Once this is in place, NAMD can be built using the CRAY-XT-g++ architecture files of the distribution.

When running NAMD on the Cray XT, the issue of the MPI-library running out of resources was frequently encountered. The resources available to the MPI-library are controlled via a number of environment settings. When running NAMD, the MPI-library typically runs out of resources during the load balancing step, when all processors need to share their performance data with rank 0. In this situation on rank 0 the MPI library requires enough memory to buffer the large number of unexpected message. If rank 0 runs out of memory, the run will fail with a clear error message explaining the problem. The error message also suggests modifications to the MPI-library's resources to resolve the problem. Setting the environment variables

```
export MPICH_UNEX_BUFFER_SIZE=100M
export MPICH_PTL_SEND_CREDITS=-1
```

inside the job submission script allowed NAMD to run successfully on the Cray XT. Most nodes of the Louhi system offer 1GB of main memory shared per core. The above environment setting gives 10% of that memory to the MPI library. This is a significant fraction of the resource.

Benchmark

The input data sets containing TCR-pMHC-CD complexes in a membrane environment was used for this study [10]. The data sets are relevant for immune response research. There were three different sets, containing one, two and nine million atoms. The basic data set with one million atoms contains four TCR-pMHC-CD complexes. The larger data sets have been generated by placing two or nine copies of this data set inside a single box.

The configurations have a step size of 2 fs. Assuming a minimum trajectory length of at least 10 ns for a scientifically meaning full simulation, these configurations require at least 5,000,000 million steps [11].

Memory footprint

The study in the very early stages of the PRACE project used NAMD version 2.6. The NAMD's memory footprint was observed to be a key obstacle when running multi million atom systems. It was not possible to run the nine million atom configuration on any platform available to the PRACE project using NAMD 2.6.

In the above context, the new possibility to reduce the memory foot print in NAMD 2.7β1 is interesting [12]. This feature is presently classified as "experimental". To reduce the memory foot print, a special NAMD executable with memory optimisation enabled needs to be built. This special executable requires compressed input files. To generate these input files, a normal NAMD executable, without memory optimisation, is used.

Utilising this feature enabled the successful execution of the 9 Million atom benchmark on a number of systems, including the Cray XT. When using the Louhi system, offering on average 1GB of memory per core, this benchmark still failed when running on 4096 cores, indicating out-of-memory errors.

To quantify the memory consumption, the CrayPat tool was used to report on the heap memory consumption. This was done for the 2 million atom benchmark and repeated for a number of environment settings suggested by the NAMD developers [12].

It was observed that the memory consumed on rank 0 is significantly different from the remaining ranks. Table 1 gives a summary of the profiling results. The table reports on the memory consumed by rank 0 and the average over the remaining ranks.

Number of tasks	Memory reduction	No Patch on Zero	Unload Zero	Footprint rank 0	Average footprint
256	No	Default	Default	1.58 GB	0.87 GB
512	No	Default	Default	1.58 GB	0.85 GB
1025	No	Default	Default	1.52 GB	0.85 GB
256	**Yes**	Default	Default	0.60 GB	0.44 GB
512	**Yes**	Default	Default	0.58 GB	0.44 GB
1025	**Yes**	Default	Default	0.60 GB	0.43 GB
256	**Yes**	**Yes**	Default	0.60 GB	0.43 GB
512	**Yes**	**Yes**	Default	0.58 GB	0.43 GB
1025	**Yes**	**Yes**	Default	0.59 GB	0.42 GB
256	**Yes**	**Yes**	**Yes**	0.56 GB	0.43 GB
512	**Yes**	**Yes**	**Yes**	0.60 GB	0.43 GB
1025	**Yes**	**Yes**	**Yes**	0.60 GB	0.42 GB

Table 1. Memory footprint for a 2 million atom simulation

The results show that enabling memory reduction has a significant effect on the memory consumption as

claimed by the developers. The other investigated NAMD settings, `noPatchesOnZero` and `ldbUnloadZero` do not yield a significant effect on the memory consumption. At least not for the benchmarks used within this project. It is also interesting to note that the memory consumption is essentially independent from the task count.

Performance

It is obvious to ask, how does the new NAMD version 2.7β1 perform in comparison to the older version 2.6. For the test system with 1 million atoms, this is shown in Figure 1 for a number of task counts. In this subsection, all measurements are for a Cray XT5 with 2.3 GHz Opteron Barcelona processors, unless otherwise noted.

Figure 1 shows that there is a substantial performance improvement when using NAMD 2.7β1. This holds in particular for the scalability of the application. The figure also shows that there is a further slight improvement to the performance when using an executable with the memory optimisation enabled.

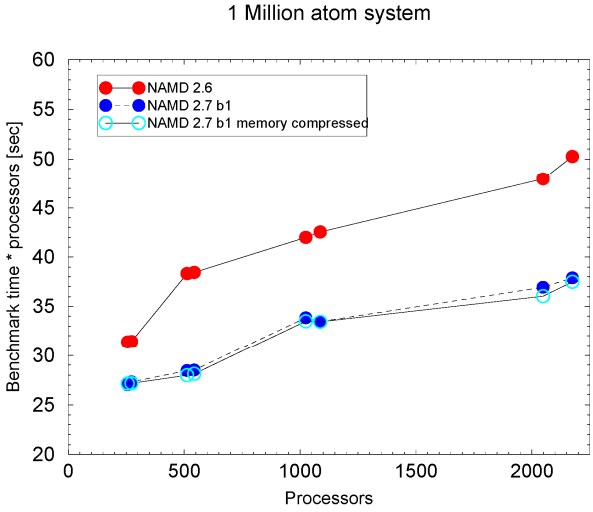

Figure 1 Performance comparison of NAMD 2.6 and 2.7β1

Since simultaneous multi threading (SMT), aka hyper threading, is not a feature on the Opteron processors available in the present Cray XT systems, the NAMD experience with SMT on the IBM Power 6 prototype is reported here. The results are shown in Figure 2 for NAMD 2.6 and 2.7β1. Figure 2 shows the performance against the physical processors. E.g. when using SMT on 512 physical processors, 1024 compute tasks were used.

The figure below shows that all NAMD versions investigated benefit from SMT. With SMT enabled, the new NAMD 2.7β1 shows a substantial improved scaling when compared to the older NAMD 2.6. NAMD responding well to SMT has been observed before for a Power5 system [13].

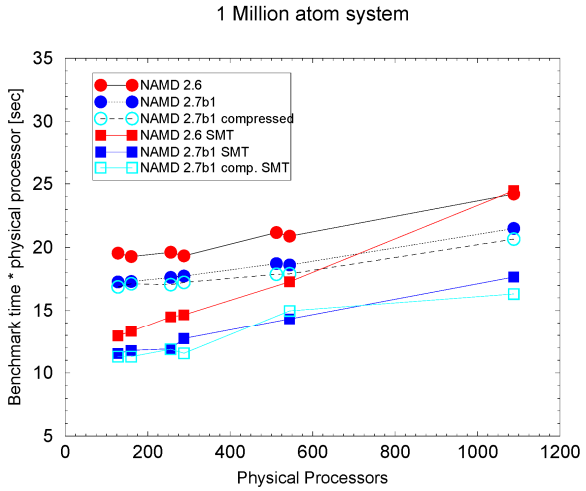

Figure 2 Effect of SMT on the IBM Power6 prototype

The next question to be discussed is the feasibility of a multi million atom simulation. Figure 3 shows an estimate, based on the NAMD benchmark time, for the wall time required for 10ns simulation. The figure shows for the Cray XT5 using 2.3 GHz Barcelona processors and the IBM Power6 system. The horizontal axis is labelled "Partition Peak", which is the product of the physical processors used and their peak performance.

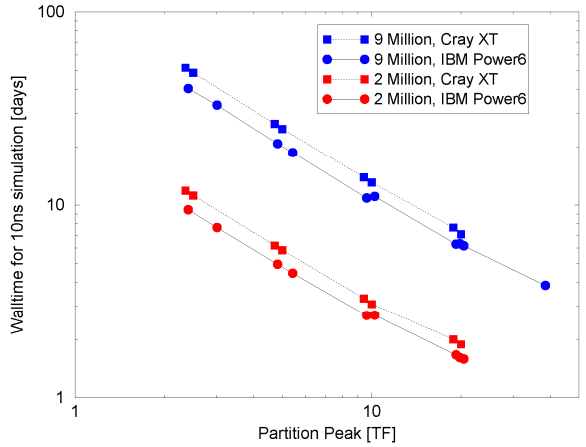

Figure 3 Feasibility of multi million atom simulations when using NAMD 2.7β1

The figure shows that when using a 20TF partition, either machine can simulate 10ns in less than 2 days wall time for the 2 million atom system and would use about a week of wall time for the 9 million atom system. If the

underlying scientific question is judged important enough the wall times involved are by no means excessive.

Compiler optimisation

On the Cray XT, the effect of the compiler flags on the NAMD performance was also investigated. This was done with gcc 4.3.3 and the performance was measured on 2.3 GHz Opteron Barcelona and 2.7 GHz Opteron Shanghai processors.

The architecture specific files for the Cray XT supplied with the NAMD 2.7b1 source suggest the following compiler options, called "default" here:

```
-O3 -ffast-math -static
-fexpensive-optimizations
-fomit-frame-pointer
```

It is important to note that for gcc 4.3.3 the option `-fexpensive-optimizations` is included at level `-O2` or higher and is therefore redundant. The following table lists the experiments and the best "Benchmark time" out of a number of re-trials. This was investigated for NAMD 2.7b1 when using a reduced memory footprint. The 2 million atom benchmark was benchmarked on 256 compute tasks:

No	Compiler options	2.3GHz proc.	2.7GHz proc.
1	`-O2`	0.204 s	0.165 s
2	`-O3`	0.203 s	0.163 s
3	`-O3 -funroll-loops`	0.196 s	0.157 s
4	`-O3 -ffast-math -static -fexpensive-optimizations -fomit-frame-pointer`	0.203 s	0.160 s
5	`-O3 -ffast-math -static -fexpensive-optimizations -fomit-frame-pointer -funroll-loops`	0.203 s	0.160 s
6	`-O3 -funroll-loops -ffast-math`	0.202 s	0.160 s
7	`-O3 -static -fexpensive-optimizations -fomit-frame-pointer -funroll-loops`	0.196 s	0.158 s

Table 2 Compiler flags Effect on NAMD performance

There was only a small dependency on the compiler options. The flags No 5, which are suggested by the NAMD developers, give already a very good performance. It was noticed that `-funroll-loops` gives a slight performance advantage while the option `-ffast-math` hinders performance. The best performance is observed for No 3 which is marginally better than the "default".

Comparing Opteron systems

It is interesting to study the effect a different communication network has on NAMD performance. There are a number of systems using the Opteron processors in the PRACE project. In addition to the Louhi system, these include the HECToR system (Cray XT4) in the UK and the Ranger system (SunBlade x6420, Infiniband) in the US. During the project the HECToR system got upgraded from 2.8GHz dual core processors to 2.3GHz quad core processors. Hence the results using both types of processors are included here. At the time of the benchmark the Ranger, Louhi and HECToR quad core system featured the same 2.3GHz Barcelona Opteron processors. The benchmarking on the Ranger system used the executable installed by the NAMD developers for general use, while on HECToR and Louhi the executables used were built by the author.

The performance on these systems is shown in Figure 4. Compared with the other Cray systems the HECToR dual core machine is slightly faster. This can be entirely explained by the differences in clock frequency. For low processor count, the performance is essentially the same on the HECToR quad core system and the Louhi system. Increasing the processor count, HECToR becomes faster than Louhi. This is not surprising. On the HECToR system (Cray XT4) each quad core processor has its own dedicated SeaStar chip for network access, while on the Louhi system (Cray XT5) there is one SeaStar serving two quad core processors.

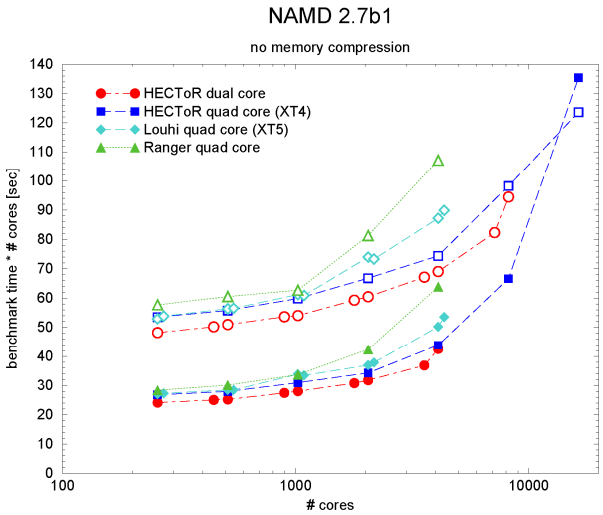

Figure 4 Performance of NAMD on different Opteron systems. The upper set of curves shows the performance for the 2 million atom benchmark, while the lower set to curves holds for the 1 million atom benchmark

When comparing the Ranger system to the quad core Cray systems, the performance is essentially the same for low processor counts, which is expected, since all systems use the same processors. The slight differences are most likely to be explained by slight differences in the build procedure. For the benchmarks used for the PRACE project, the Ranger system does not scale as well as the Cray systems. This is in contrast to the findings of [14]. In [14] for a different benchmark configuration of similar size (1 million atoms) the NAMD performance on Ranger and Jaguar (Cray XT4, 2.1 GHz quad core Opteron) is compared. Better scalability was observed on Ranger. Understanding the reasons for this difference between the investigation by the code developer and the present study would be interesting.

Comparing PRACE prototypes

Figures 5 shows the comparison on the relative efficiency of NAMD for the one million atoms benchmark on the PRACE prototypes mentioned in the introduction of this paper. Figure 6 shows the comparison for the nine million atoms benchmark. The performance is compared against the peak performance of the partition used (product of core-count and peak performance of the core). Due to memory limitations there are no results on the BlueGene/P for the nine million atom benchmark and also there is no data point for the Cray XT5 (Barcelona) for a 40 TFlop partition due to this. This illustrates, while the memory reduction is helpful, it does not fully overcome the problems. Excluding the Sun x86 Cluster, one notices a difference in performance between the prototypes for smaller partition sizes. However for larger partition sizes these differences are minimal and the performance becomes effectively independent from the prototype chosen.

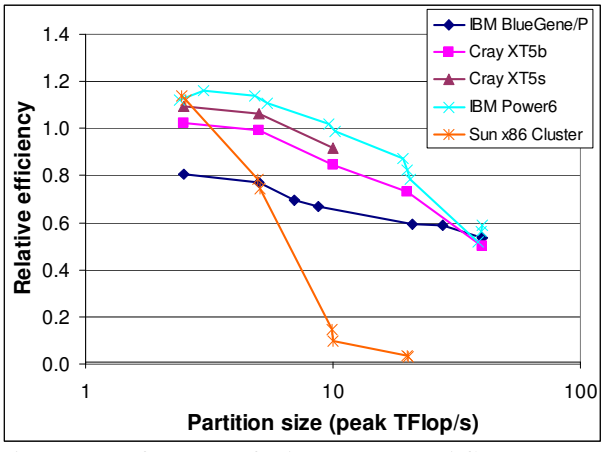

Figure 5 Performance of NAMD on the PRACE Prototypes for the 1 million atoms Benchmark

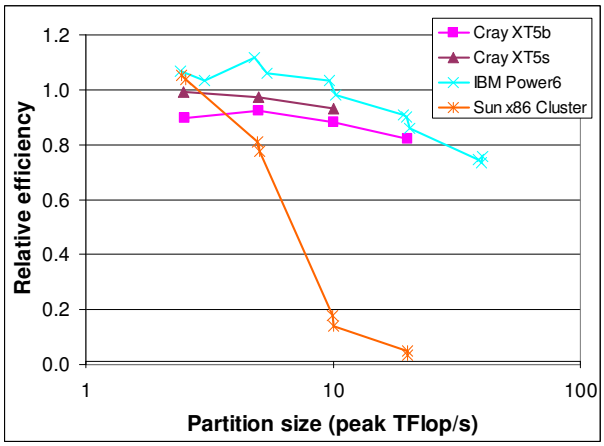

Figure 6 Performance of NAMD on the PRACE Prototypes for the 9 million atoms Benchmark

On the Sun x86 cluster the performance is good for small partition sizes, while scalability on this prototype is poor. The reasons are not understood. When these tests were performed, the architecture was newly installed and might not have been tuned properly or something might have been broken.

3. HELIUM

Overview

The application HELIUM uses time-dependent solutions of the full-dimensional Schrödinger equation to simulate the interaction between an intense linearly-polarized laser pulse and a Helium atom [15]. It is developed by the Queen's University Belfast and has access restrictions.

The HELIUM source code is written in a single Fortran 90 file with more than 14000 lines. It uses MPI for the parallelising implementation. In the HELIUM code, the total work in a whole grid space is divided into a set of blocks, which are distributed onto the MPI processes. Each process will operate on the square region of the assigned block space. For a fixed total problem size, there are multiple decomposition approaches to divide the whole grid space into blocks by changing the block size and the block number in each dimension.

There are several parameters in the source code to control the simulation conditions and the execution behaviour. The total problem size, block size, block count and core number required for the execution can all be set by a number of parameters in the source code prior to compilation. HELIUM will write out results once every several time steps and the output frequency can also be specified in the source code with particular parameters. Changing the parameters and re-building the HELIUM

executable will produce different HELIUM test cases suitable for benchmarking.

The HELIUM test case with a fixed total problem size of 1540 grid units was used for the investigations during the PRACE Preparatory Phase. The total number of time steps was set to 80 and the writing operation was set to be executed once every 20 time steps.

Porting HELIUM to Cray XT5

Porting HELIUM to the Cray XT5 prototype Louhi is relatively straightforward as the source code is only one single file and no specific libraries or environments are required for the compilation and execution. Both the PGI Fortran90 compiler and the PathScale Fortran90 compiler can be used directly for building small HELIUM test cases. However, when using the PGI compiler one encounters a reallocation limit compiling issue when building the test cases with medium or large problem size, so all the compiling on Louhi used the PathScale Fortran90 compiler. The default flag used for porting was: `-O2`.

There were several modifications needed for the original version code to pass the syntax check applied by the PathScale compiler. These changes have been merged to the latest version code by the developer.

When running the HELIUM test cases, it is important to use proper core numbers to run the test cases. The core numbers required for execution is depended on the parameters of total problem size and block number in the source code. Therefore not all the core numbers can be used for benchmarking the selected test cases. A wrong core number will lead to an execution failure.

The HELIUM execution usually consumes large amounts of memory. Overcoming the memory limitations is a key issue when porting HELIUM. The memory required by the test cases can be roughly estimated based on some parameters in the source code, including the block size. For the test case with a fixed problem size, the required memory size could be varying when using different block decomposition approaches. However, there is no guarantee for the upper limit by the estimation, so it is very difficult to predict the total memory required. The execution freezes or fails altogether, if the system runs out of memory, even when the test case executables can be built successfully. Louhi nodes offer either 8 GB or 16 GB of memory, shared between the 8 cores of the node. When using proper block decompositions to run the HELIUM test cases, the executions can generally be successful on fully populated nodes. Utilising only four or two task per 8-way node can sometimes help to solve the memory limitation issues, but the delivered performance was usually very poor.

Compiler optimisation for HELIUM on Cray XT5

The effect of compiler flags on the HELIUM performance was investigated on Louhi. For this the test case with a fixed problem size of 1540 grid units was used. The measurements were performed on the 2.3 GHz Barcelona processors using either a total of 630 cores or of 1540 cores. This investigation is based on the PathScale compiler options only.

The flag `-O3` and `-OPT:Ofast` were applied as the starting point of the compiling optimisation. `-O3` is the highest optimisation level of the PathScale compile. `-OPT:Ofast` is another common used performance tuning option for the PathScale compiling. It was effective when using the two flags together to speed the code execution without breaking the code results correctness.

The CrayPat tool was used to understand the routine expenses in the source code. Based on the profiling results on Louhi, the most expensive routines all have large calculation loops. PathScale compiler provides several loop related sub-options for the flag `-OPT` and a specific loop nested optimiser flag `-LNO:<suboptions>`, to help apply the loop optimisation techniques without code changing by hand [6]. The following table shows the loop optimisation options which were investigated and tested for the HELIUM test case on Louhi.

Flags Usage	Effects
`-LNO:fusion=2`	The highest-level loop fusion. It can help change the temporal locality.
`-LNO:fission=0`	Loop fission with the lowest level as it will cancel out the effect of loop fusion if switched on.
`-OPT: unroll_analysis =ON`	Turn on the unroll analysis.
`-LNO: full_unroll_size =2000`	Loop unrolling with the default unrolling loop size 2000. It can increase the loop body size and give more scope for better schedules and reduce branch frequency while keeping the code readability.
`-LNO:simd=2`	The highest-level loop vectorisation. Use SIMD instructions to improve the calculation performance.
`-OPT:pad_common =ON`	To reduce conflict cache misses. It was not applied in the end as the effect was not as good as expected.

Table 3 The PathScale compiler flags used for HELIUM loop optimisation on Cray XT5 (Barcelona)

The final flags delivering the best observed performance for the HELIUM test case with problem size of 1540 grid units on Louhi are:

```
-O3 -OPT:Ofast:unroll_analysis=ON
-LNO:fusion=2:fission=0
:full_unroll_size=2000:simd=2
-LIST:all_options=ON
```

Table 4 shows the HELIUM performance improvement using these optimised compiler flags.

Cores	-O2	-O3 -OPT:Ofast	-O3 -OPT:Ofast :unroll_analysis=ON -LNO:fusion=2:fission=0 :full_unroll_size=2000 :simd=2 -LIST:all_options=ON
630	936 s	731 s	729 s
1540	338 s	316 s	312 s

Table 4 HELIUM performance improvement with compiling optimisation on Cray XT5 (Barcelona)

Reducing MPI cost

The CrayPat MPI profile showed that the MPI communications have more and more impact on the code performance when utilising a large number cores. To reduce the communications cost, the debugging routine `Test_MPI` was removed. The routine `Test_MPI` called several MPI communication routines at the code initialisation stage for the testing purposes only, which was unnecessary for the real code execution.

The effect of removing the MPI debugging routine depends a lot on the system, probably due to the different MPI implementations. The performance improvement was not quite obvious on Cray XT5, compared with the effect on the IBM Power 6 prototype. Table 5 and Table 6 are the performance comparison on the Cray XT5 system and the IBM Power 6 system before and after removing the debugging MPI routine.

Cores	Before removing Test_MPI	After removing Test_MPI
630	729 s	729 s
1540	312 s	311 s

Table 5 Performance comparison before and after removing Test_MPI in HELIUM on Cray XT5 (Barcelona)

Cores	Before removing Test_MPI	After removing Test_MPI
630	714 s	712 s
1540	319 s	309 s

Table 6 Performance comparison before and after removing Test_MPI in HELIUM on IBM Power6

Merging loops for HELIUM on Cray XT5

Based on the CrayPat routine profiling on Louhi, there were four very expensive routines in the original code, including `Incr_with_1st_Deriv_op_in_R1`, `Incr_with_1st_Deriv_op_in_R2`, `Incr_with_2nd_Deriv_in_R1`, and `Incr_with_2nd_Deriv_in_R2`. It was founded in these routines that they all have some heavy calculations which were split into several loops with the same boundaries. This was not very efficient for the loop iterations. Besides, the loops didn't go through in a proper order, which could cause more cache misses. Therefore, such loops were merged together with a proper iteration order to improve the HELIUM performance on the Cray XT5 prototype system.

The following tables are the performance comparison before and after merging the loops. All the tests were taken after removing the MPI debugging routine in the code. It can be seen from Table 7 and Table 8 that after merging the loops, the HELIUM performance was improved effectively on Louhi and the cache misses reduced as well.

Cores	Before merging loops	After merging loops
630	729 s	723 s
1540	311 s	302 s

Table 7 Performance comparison before and after merging loops in HELIUM on Cray XT5 (Barcelona)

Cores	L1 Cache misses		L2 Cache misses	
	Before merging loops	After merging loops	Before merging loops	After merging loops
630	6.62E+09	2.30E+09	2.97E+09	7.05E+08
1540	1.69E+08	1.50E+08	1.63E+08	9.02E+07

Table 8 Cache misses comparison before and after merging loops in HELIUM on Cray XT5 (Barcelona)

Performance and scalability

Figure 7 shows the performance of HELIUM on Louhi before and after applying all the optimisations discussed above. The original scaling results are represented by the green line and the optimised new results are shown by the red line. The results were measured based on the test case with a fixed problem size of 1540 grid units on the 2.3 GHz Opteron Barcelona processors. It can be seen from Figure 7 that the HELIUM test case scaled from 630 cores up to 2485 cores. For all core counts the performance is improved after applying the optimisation techniques.

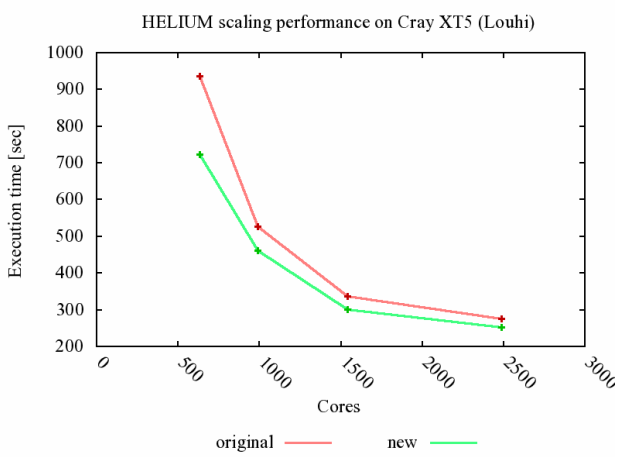

Figure 7 HELIUM performance before and after optimisation on Cray XT5 (Barcelona)

Figure 8 is the corresponding cost plot of Figure 7. The cost represents the product of execution time and the core number. A horizontal cost curve implies a linear scaling. Figure 8 shows when using less than 1540 cores, the HELIUM test case had a super-linear/linear scaling. However, when using much larger core number to run the code, the scalability tailed off significantly, which is probably mainly due to the heavy MPI overheads. After applying the optimisations, the cost of HELIUM reduced but there was no obvious improvement for the scalability tail-off.

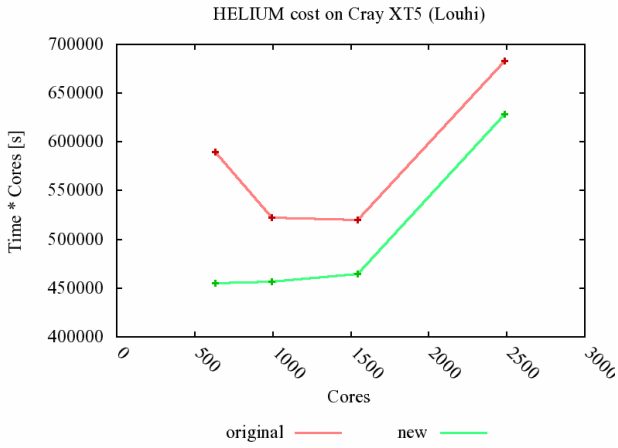

Figure 8 HELIUM scaling cost before and after optimisation on the Cray XT5 prototype system Louhi (Barcelona)

Comparison on PRACE prototype systems

Figure 9 shows the relative efficiency comparison on multiple PRACE prototypes for HELIUM, which is similar with the investigation for the NAMD research discussed in the Section 2. The performance of HELIUM is compared against the peak performance of the partition. The test case used for the comparison has a fixed problem size of 1540 grid units.

It can be seen from Figure 9 that for the HELIUM application test case, both the Cray XT5 system and the IBM BlueGene/P system scaled well, but the Cray XT5 was between 1.5 and 1.7 times as efficient as the IBM BlueGene/P. The performance on the IBM Power6 and the Sun x86 cluster was good when using small partition size, but the scalability on these two prototype systems was poor.

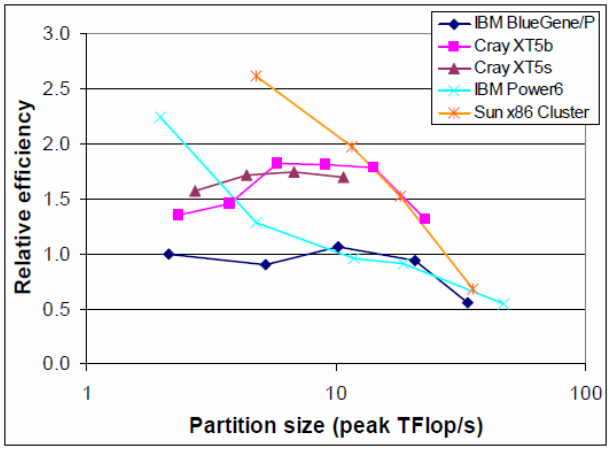

Figure 9 Performance of HELIUM test case (problem size 1540) on PRACE prototype systems

4. Conclusion

In the WP6 of PRACE project Preparatory Phase, EPCC was responsible for the applications enabling work of NAMD and HELIUM on future petaflop/s systems. Both of the applications were ported to a number of PRACE prototypes, including the Cray XT5 prototype, Louhi. The scalability of the two applications was investigated and the optimisation strategies for petascaling were implemented based on the profiling results, which delivered better performance and scalability for the two applications. In general, both applications are fit for the future petaflop/s systems of the forth coming PRACE Implementation Phase.

NAMD has a good scalability on the Cray XT systems. It scaled up to more than 4000 cores on Louhi with the 1 million and 2 million atom benchmark on the 2.3GHz Barcelona Opteron processors and can scaled using many thousands of cores on other Cray XT systems. The new version NAMD 2.7β1 was ported successfully to Louhi, which delivered a substantially improved performance compared with the older version NAMD 2.6.

The code performance has only a small dependency on the compiler options. Using the new "experimental" NAMD feature to reduce the memory consumption of the code has enabled the simulation of multi-million atom systems on a Petaflop/s system. The wall clock time required for the simulation of such a system with a reasonably sized trajectory is manageable on a modern HPC platform.

HELIUM is relatively straightforward to be ported to the Cray XT5, but it requires proper settings in the source code for the problem size, decomposition approach as well as the core number. The memory consumption of HELIUM is large. This becomes a key issue when porting large problem size test cases to a compute platform offering limited amounts of memory per compute core. The performance of HELIUM was improved efficiently by the PathScale compiler options, especially by using the loop optimisation flags. This is mainly because that HELIUM has several expensive routines with large loop calculations. Merging such large loops with the same boundaries in a proper order can also improve the HELIUM performance, which leads to a better cache usage on the Cray XT5. Compared with other PRACE prototypes, HELIUM delivers good scalability and performs efficiently on the Cray XT5 prototype system Louhi.

Acknowledgments

The authors would like to thank the PRACE partner contributors Martin Polak (ICA, Johannes Kepler University Linz, Austria), Paschalis Korosoglou (AUTH, Greece) and Andrew Sunderland (STFC Daresbury Laboratory, UK) for their contributions.

The authors would also like to thank their colleagues, the applications developers and the vendor staff for all their great support and help.

About the Authors

Xu Guo is an Application Consultant of EPCC, The University of Edinburgh. She can be reached at EPCC, University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, UK. Email: xguo@epcc.ed.ac.uk.

Joachim Hein is a Computing Architect at EPCC, The University of Edinburgh. He also works as a researcher at the Centre for Mathematical Sciences at Lund University. Email: j.hein@ed.ac.uk.

Reference

[1] *About PRACE.* Online at http://www.prace-project.eu (Referenced on 30/04/2010).

[2] *PRACE principal and general partners*. Online at http://www.prace-project.eu/about-prace/partners (Referenced on 30/04/2010).

[3] *PRACE brochure 2009*. Online at http://www.prace-project.eu/documents/PRACE_brochure_09.pdf (Referenced on 30/04/2010).

[4] *Software enabling for Petaflop/s systems*. Online at http://www.prace-project.eu/activities/work-package-6 (Referenced on 30/04/2010).

[5] *Press release: PRACE Benchmark Suite Finalised.* Online at http://www.prace-project.eu/documents/press-releases-pdfs/prace_benchmark_pr.pdf (Released on 15/02/2010; Referenced on 30/04/2010).

[6] *Louhi User's Guide, the 2nd Edition*. Online at http://www.csc.fi/english/pages/louhi_guide (Referenced on 30/04/2010).

[7] *NAMD2: Greater Scalability for Parallel Molecular Dynamics*, L. Kalé, et al., Journal of Computational Physics **151**, 283 (1999)

[8] *Scalable Molecular Dynamics with NAMD*, J. Phillips, et al., Journal of Computational Chemistry **26**, 1781 (2005)

[9] *Charm++: Parallel Programming with Message-Driven Objects*, L. Kalé, S.Krishnan, in: *Parallel Programming using C++*, by G.V. Wilson and P. Lu. MIT Press, 175 (1996)

[10] Peter Coveney, Shunzhou Wan, private communication

[11] Peter Coveney, private communication

[12] http://www.ks.uiuc.edu/Research/namd/wiki/index.cgi?NamdMemoryReduction

[13] *An Investigation of Simultaneous Multithreading on HPCx*, Alan Gray et al., HPCx technical report, `http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0604.pdf`

[14] *Understanding Application Performance on Three Predominant Supercomputer Architectures: Intrepid, Ranger and Jaguar, using Micro-benchmark*, Abhinav Bhatele et al., Preprint, PPL Paper: 10-03, `http://charm.cs.illinois.edu/papers/PerfCompIJHPCA10.shtml`

[15] *Numerical integration of the time-dependent Schrödinger equation for laser-driven helium*, Edward S. Smyth, Jonathan S. Parker, K.T. Taylor, Computer Physics Communications **114** (1998) 1-14.