D4.1.1 – Overview of major limiting factors of existing algorithms and libraries

WP4: Algorithms and libraries

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:	6
Submission date:	31/03/2012
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	High Performance Computing Centre Stuttgart (HLRS) University of Stuttgart
Version:	1.0
Status	Final
Author(s):	Stephen P Booth (UEDIN), Dmitry Khabi (HLRS), Gregor Matura (DLR) Cristoph Niethammer (HLRS) Harvey Richardson (CRAY UK)
Reviewer(s)	Alan Gray (UEDIN), Tobias Hilbrich (TUD)

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	30.01.2012	First version of the deliverable	Dmitry Khabi (HLRS)
0.2	20.02.2012	Added section about FFT	Stephen P Booth (UEDIN)
0.3	23.02.2012	Added Introduction	Dmitry Khabi (HLRS)
0.4	01.03.2012	Comments and some changes	Harvey Richardson (CRAY UK)
0.5	07.03.2012	Added short description of Jacobi, CG, Multigrid, H-Matrix and svn repository	Dmitry Khabi (HLRS)
0.6	09.03.2012	Added section Conclusions, collective operations and future work	Harvey Richardson (CRAY UK), Cristoph Niethammer (HLRS), Dmitry Khabi (HLRS)
0.7	15.03.2012	Changes due to review	Dmitry Khabi (HLRS)
0.8	26.03.2012	Changes due to review	Harvey Richardson (CRAY UK)

Table of Contents

Index of Figures

Index of Tables

1 Executive Summary

In this deliverable, we give an overview of the main external libraries that are of importance for the CRESTA Co-design applications. This document presents the state of the art for the use of external libraries in Co-design applications, and presents new methods for their use. The problems faced today and in the future are described. A list and associated brief explanation are provided for algorithms that are currently being used or are under development. Some of the implementations have already been tested on most modern platforms. For this purpose, Cray XE6 and NEC Nehalem platforms have been utilised. The deliverable shows how the algorithms can be described mathematically to study their suitability for future exascale systems. Statistics have been collected for the verification of such models, which can be expanded at any time, depending on the requirements of future models. Finally there is a description of the CRESTA SVN repository for WP4, which should unite the software that we are going to use and develop in the CRESTA project.

2 Introduction

Our small but detailed survey of the European HPC user communities[1] has shown that many of the applications tend to use their own implementations of parallel computation in preference to using any external libraries. For this there are a number of reasons: additional difficulties in linking external libraries, specific features of the problems to be solved, some difficulties in the control of the external solvers, the need to reformat data for export/import to an external library, and so on. But the requirements for the algorithms will be dramatically increased on going to the new scales of simulated tasks (increasing of the computational area, the amount and complexity of processors and their interconnectivity, limitation on power consumption). There's a lot of doubt that those solutions that more or less fit today will be able to give satisfactory results even in the near future. This was also confirmed by the European HPC user communities. So it is not advisable to develop in parallel the same methods on different systems, and so disperse the limited resources.

The only exceptions to this trend for our Co-design applications are ELMFIRE that uses PETsC (the software library for domain decomposition and solving system of linear equations) and HemeLB that uses ParMetis (used for domain decomposition and graph partitioning). This is one of the main reasons for our choice of the PETsC library for our initial tests. The application HemeLB is a Lattice Boltzmann code so it does not use any FFT libraries or linear system solver libraries. The library ParMetis is used by HemeLB during the initialization phase in order to optimize the domain decomposition. Also based on the experience of using this library in the pre-processing phase, it seems that ParMetis will not be the weak link in this exascale computation.

To identify the external libraries for NEK5000 we need to work more closely with its users within CRESTA. This is due to the specific assembly of its matrices for the systems of linear equations. The first contacts and specific arrangements towards this goal have been agreed upon at the CRESTA collaboration meetings in Edinburgh and Cologne.

The Spectral Transformations used within the IFS and GROMACS systems requires global communication. And this seems to be a weak link for exascale computing. Therefore, we consider a model for the FFT computation, with an emphasis on global communications. New developments in the field of global communications should be especially tested in this area. To do this, we have also developed some tests for FFTs. The integration of these tests into the test environment of WP4 is in progress.

The application OpenFOAM is very complex in terms of the Software Engineering and uses innumerable C++ classes. Their definition is also hidden with the help of templates. We have identified the classes that we will use to connect an external solver for the systems of linear equations. Currently, work is underway to release the matrix data suitable for this purpose.

The table below highlights whether the Co-design application requires a new approach towards HPC computation.

[1] A recent survey carried out across the CRESTA co-design applications.

Application	Requirement for a new approach
HemeLB	NO
ELMFIRE	YES (iterative methods, Multigrid)
IFS	YES (FFT)
GROMACS	YES (FFT)
OpenFOAM	YES (iterative methods, Multigrid)
NEK5000	YES (iterative methods, Multigrid)

Table 1 - Requirement of the Co-design applications

All CRESTA Co-design applications use libraries such as BLAS and LAPACK. This is unsurprising as this has become standard in HPC. With their help, local calculations are made for standard tasks, e.g.: scaling of vectors, dense matrix multiplication, solving small systems of linear equations. There are a large number of high-performance implementations that are partially developed by processor vendors. In this first task for WP4 we didn't examine these libraries, as they are not the factor that prevents applications from being highly effective. On the contrary, their use is one of the essential factors for their success. It is of coursed necessary to consider this. But first we need to study other factors which limit the effectiveness of the applications. Initial progress has been made towards this and this document presents this progress.

2.1 Purpose

The purpose of this deliverable and our future work is as follows:

- Overview of the most important algorithms for the solving of large systems of linear equations.
- Create a suitable benchmark system to test the existing external libraries and to support the development of new algorithms.
- Analyse the most prominent libraries for their suitability to solve large systems of linear equations.
- Deliver statistical data in order to develop and evaluate new models (including the model for FFTs presented in this deliverable) for exascale computation.
- Select the best algorithms and their implementation for the MPI collective operations that occur in the CRESTA Co-design applications.
- Creating an external solver for a system of linear equations, and the mechanism of its connection to a variety of CRESTA's Co-design applications for effective distribution of tasks on exascale systems.

2.2 Glossary of Acronyms

AMG	Algebraic Multigrid
CG	Conjugate Gradient Method
CrayPat	Cray Performance Analysis Tools
CRS or CSR	Compressed Row Storage format
DFT	Discrete Fourier Transform
FE	Finite-Element
FFT	Fast Fourier Transforms
H-MATRIX	Hierarchical Matrix
LU	LU decomposition
MVM	Matrix Vector Multiplication
PETsC	Portable, Extensible Toolkit for Scientific Computation
SVN	Subversion

3 Simulation Problems and Algorithms for their Solution

Solving the issues of biomolecular systems, fusion energy, the virtual physiological human, numerical weather prediction and engineering plays an increasing role in modern life. In order to meet the growing demands on these process simulations, the computational program may need to use new approaches. When scientists or engineers simulate a continuous event, for example the flow of a fluid through a pipe, they impose a grid over the area of interest and compute the relevant parameters in the nodes of the grid. This often leads – alongside the definition of the parameters on the boundary grid nodes – to a large system of linear equations $A\vec{x} = \vec{b}$. The matrix A describes the relationships between the parameters (e.g. the velocity of the flow in the grid point), the vector $\vec{x}$ is the unknown parameter and the vector $\vec{b}$ is the given input (e.g. the velocity of the flow at the beginning and end of the pipe). One of the current bottlenecks in the scientific computation is the solving of these very large systems of linear equations. This relates to square matrices A of size 10^8 x 10^8 and higher.

In the next sections we introduce some important algorithms for HPC computation. Some of them are widely used and others just beginning to be utilised by HPC applications. Detailed descriptions of the algorithms can be found in many books and on the Internet. We also provide some references to, in our opinion, the most informative and modern resources. Following this, we describe our testing and software development for parallel computing.

3.1 Overview of the Algorithms

For many centuries, scientists developed new methods for solving the simple equation $A\vec{x} = \vec{b}$ fast and reliably. Carl Friedrich Gauss developed the Gaussian elimination technique around the 1800's. This method has been used effective until today. Its advantages include the techniques guarantee to solve the problem. It makes this algorithm with some modifications (e.g. pivoting) one of the most frequently used sequential computations for solving systems of linear equations of small order (with the number of equitation less than 10^4). Unfortunately (or fortunately for computer scientists), its degree of complexity ($O(n^3)$) and inability to be parallelized makes it impractical for solving large systems of linear equations. Even if we could determine how to parallelize the algorithm with one hundred percent efficiency, it is unlikely that it will be helpful for us. In order to solve the system with 10^8 of linear equations, we need to perform $10^{3x8} \equiv 10^{24}$ operations. One exascale computer will compute 10^{18} operations in a second. We would still require more than 12 days to compute the solution with such a computer.

The next step in the development of algorithms for solving systems of linear equations is the use of iterative methods. One of the simplest and oldest of these algorithms is the Jacobi method. Unlike Gaussian elimination, the Jacobi method computes an approximate solution. Unfortunately the Jacobi method typically converges very slowly. But some of its beneficial properties make it attractive for use in modern algorithms: the Multigrid method (so-called Jacobi relaxation step (1)). The most prominent iterative method for solving sparse systems of linear equations is the Conjugate Gradient Method. It was developed around the 1950's by Cornelius Lanczos and Walter Arnoldi. There are many different modifications of the algorithm (Bi-CG, GMRES...), so that almost any system of linear equations, if at all possible, can be approximately solved. One of the main advantages of this method is that it simple to parallelize, which greatly reduces the duration of a single iteration. Unfortunately, with the number of iterations required to solve the system of linear equations, it is often an insurmountable problem. Hence Krylov space methods are typically used in conjunction with a preconditioner. The new theory of hierarchical matrices (2) promises to be one of the most interesting ways of finding an effective preconditioner.

3.1.1 Jacobi

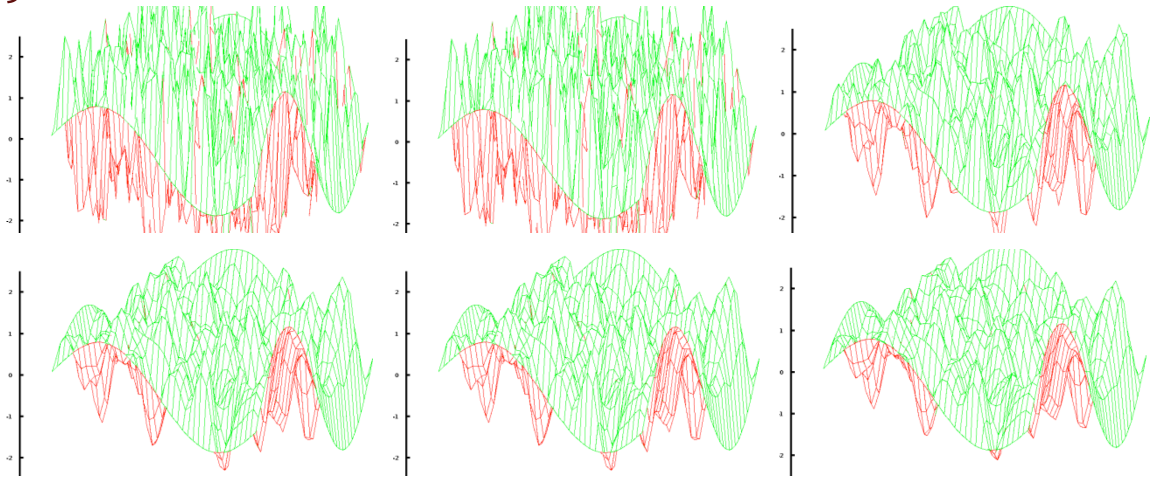


Figure 1 - The first 6 iterations of the Jacobi method (Poisson's equation)

Figure 1 shows the approximate solutions to Poisson's equation after the first six iterations calculated by the Jacobi method. The finer the mesh the more iterations needed to achieve a sufficient accuracy of the solution (pictured below in the Figure 3 for the Multigrid method). But the high error frequencies are attenuated within a few iterations.

3.1.2 Conjugate Gradient

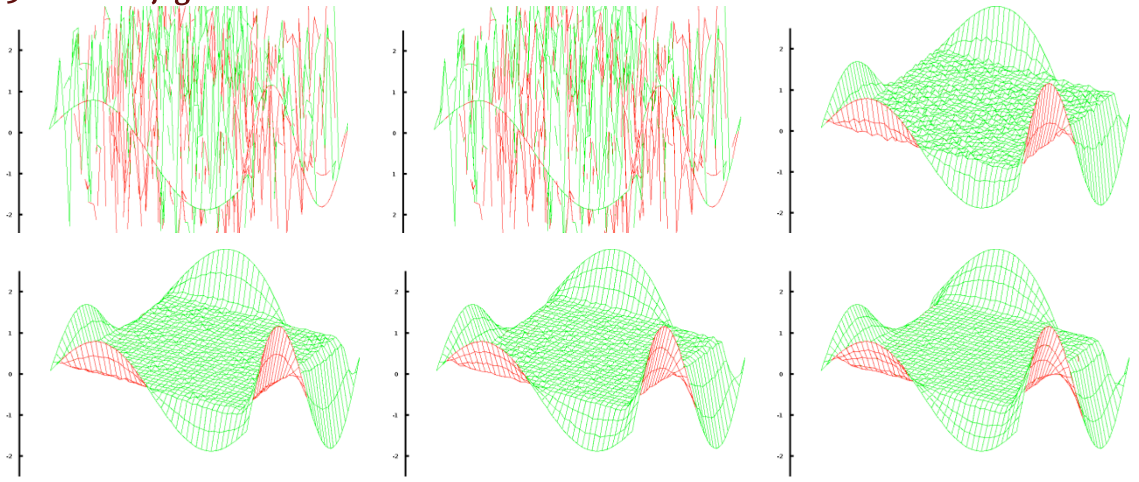


Figure 2 - The first 6 iterations of the CG method (Poisson's equation)

The CG method converges faster than the Jacobi method. The reduction of the error works from the edge to the centre (see Figure 2).

3.1.3 Multigrid

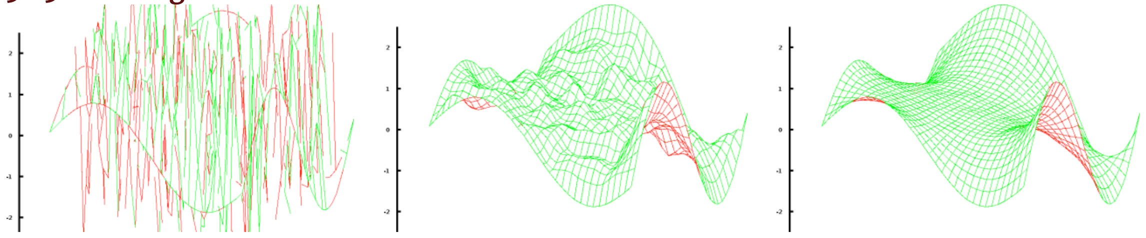


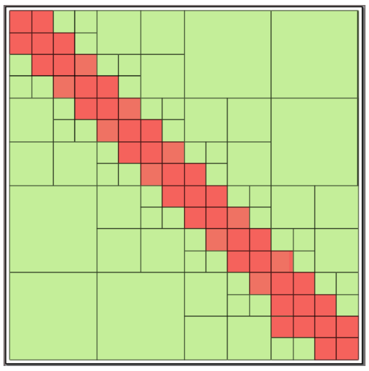
Figure 3 - The first 3 iterations of the algebraic Multigrid method (Poisson's equation)

This method converges very quickly. The error reduction is independent of the mesh size. The error is reduced by one order per iteration. The method is much more complicated than Jacobi and CG. For more details about the Multigrid method see one of the most prominent books in this area: (1).

Many more details and source code for these methods can be found in various books and on the Internet. One of the latest detailed overviews was done in lectures in the University Stuttgart, see (3). The images were also made with the help of various source codes, which can be found in these University of Stuttgart lectures.

3.1.4 Hierarchical Matrix

The main idea in the "H-Matrix"(2) theory lies in the fact that the matrix A can be expressed in approximate form. An approximation is acceptable as a discretization of differential equations by itself leads to some errors: it is called discretization error. For the approximation, the matrix is divided into many blocks of various sizes. Some of these blocks can then be represented as low-rank matrices of rank *k*. A low-rank matrix has many fewer elements than the exact matrix (if *k* much less as number of rows). One of the major challenges is to divide the matrix, so that the blocks are large enough, and the parameter *k* (rank of the block matrix) is small. In this case, it is possible to construct a good preconditioner (like LU) with almost linear complexity. On the other hand there is a problem of data distribution and load balancing. There are several libraries for solving these systems of linear equations using H-Matrix on shared memory systems (H-Lib^{pro} is the one of the best implementation of it (4)). Currently, work is underway to develop a library that uses MPI. In the CRESTA project, we also want to try to apply this new theory to solve systems of linear equations.



🟥 Dense block matrix

🟩 Low-rank block matrix

Figure 4 - Example of the H-Matrix

3.1.5 FFTs

Fast Fourier Transforms are the generic name for a class of algorithms used for computing Fourier transforms (technically Fourier transforms are continuous functions and FFTs actually calculate the discrete approximation to the Fourier transforms).

The Discrete Fourier Transform can be defined as follows:

$$DFT(k) = \sum_{x=0}^{N-1} F(x)e^{-i2\pi k x}$$

This is defined for all $k = 0, \frac{1}{N}, \frac{2}{N}, \cdots \frac{(N-1)}{N}$

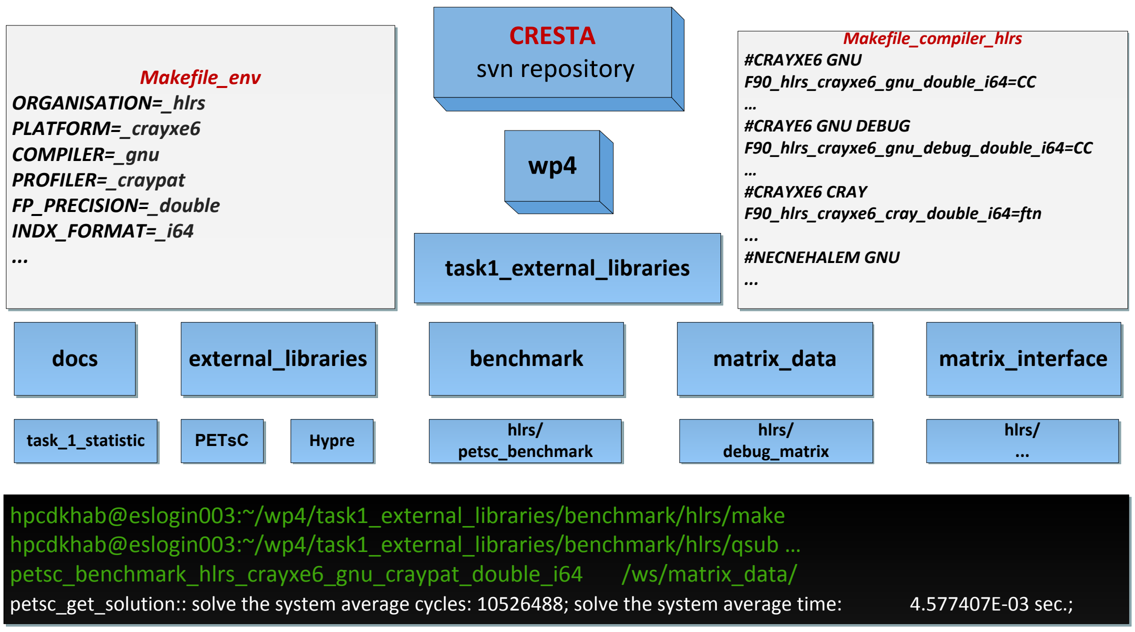
Though a naïve implementation of the DFT would require O(N^2) operations the FFT algorithm is a very efficient recursive algorithm with complexity O(N log(N)). Many computational problems contain particular calculations that are significantly easier to compute using either real or spectral space data representations. This can be a very large saving, for example replacing an expensive iterative solver with a local calculation. Unfortunately many applications contain a mixture of calculations with different optimal data representations but the highly efficient nature of the FFT algorithms make it feasible to change data representations many times during the execution of the application.

4 CRESTA SVN repository WP4/task1_external_libraries

In order to test the existing libraries for solving the systems of linear equations of the CRESTA Co-design applications we require a lot of different modules and data sets, here some of them:

- External Libraries (PETsC, Hypre, Trilinos, …)
- Data for the tests
- Programs to generate the test data
- Modules to read and distribute the data
- Benchmarks
- Environment settings for different platforms

These modules will be needed throughout the project to develop our own system, for solving systems of linear equations. For these reasons we have created the CRESTA SVN repository*WP4/task1* in which we manage the above-mentioned modules. During the project "CRESTA" we will develop this further. This is, among other things, the basis for the work of the “linear solver” co-design team, which brings together teams from WP4, WP5, WP6 and WP2. More information for this is located in CRESTA’s BSCW under the directory "Co-design teams". Figure 5 illustrates the structure of the repository. In addition we have provided support for semi-automatic compilation and integration of different modules on HLRS platforms. The flexibility of this test system allows this support to be extended to other platforms and external libraries that are used in the project.



Makefile_env
```
ORGANISATION=_hlrs
PLATFORM=_crayxe6
COMPILER=_gnu
PROFILER=_craypat
FP_PRECISION=_double
INDX_FORMAT=_i64
...
```

Makefile_compiler_hlrs
```
#CRAYXE6 GNU
F90_hlrs_crayxe6_gnu_double_i64=CC
...
#CRAYE6 GNU DEBUG
F90_hlrs_crayxe6_gnu_debug_double_i64=CC
...
#CRAYXE6 CRAY
F90_hlrs_crayxe6_cray_double_i64=ftn
...
#NECNEHALEM GNU
...
```

```
hpcdkhab@eslogin003:~/wp4/task1_external_libraries/benchmark/hlrs/make
hpcdkhab@eslogin003:~/wp4/task1_external_libraries/benchmark/hlrs/qsub ...
petsc_benchmark_hlrs_crayxe6_gnu_craypat_double_i64      /ws/matrix_data/
petsc_get_solution:: solve the system average cycles: 10526488; solve the system average time:          4.577407E-03 sec.;
```

Figure 5 - CRESTA SVN repository for the analyse and tests of external libraries

In the directory "*wp4/task1_external_libraries/docs/task1_statistic*" are the statistics that we have collected during initial testing. Some of the data was used in this document. This data will also be used in the next task of WP4 for the development of a "Prediction model for limiting factors for hardware and algorithm libraries”.

5 External Libraries and their Application

5.1 Iterative Methods

5.1.1 Test environment for external libraries

As a representative library of iterative methods we have selected the library PETsC (5). PETsC is written in C ++ and Fortran and uses MPI for all message-passing communication. The core part of the library deals with solving systems of linear equations. It provides a highly scalable implementation of the CG-method with some simple preconditioners. We analysed benchmark results on the examples for matrix size of 10^8x10^8 to determine at first the most important challenges of an exascale computation. For some tests, we've used matrix size of 10^5x10^5, which was assembled for the same simulation problem but for a smaller domain. Generating matrices of different sizes based on the simulation problems of the Co-design applications was not possible within this timescale, as the assembly of the matrix is hard bound to the code of the applications. One of our next tasks is to solve this problem (one possibility is to use the I/O server architecture, see below).

CG embodies all the important sub-operations used in iterative methods. The library can be compiled for single-, double- and long-double-precision computation. The configuration of the solver can be made directly in the code or by using a configuration file. We used the later for our experiments. In the following table we present the chosen parameters for solving the system of linear equations.

```
-ksp_type cg # Conjugate Gradient Method

-pc_typeJacobi # preconditioner

-ksp_rtol 1.e-12 # residual norm relative to the norm of the right hand (b)

-ksp_atol 1.e-52 # absolute size of the residual norm

-ksp_divtol 100000 # divergence

-ksp_max_it 1000 # maximum number of iterations

-ksp_converged_reason # gets the reason the KSP iteration was stopped
```

Table 2 - The configurations file for the solver of the system of linear equations

The KSP is the object for access to the Krylov subspace methods. In the configuration above it was set to CG. The Jacobi preconditioner was selected for our benchmark program. For the system of linear equations we used it was the only possible candidate implemented in this library. One of the features of the matrix that we used was it was not a block matrix. This is due to the fact that during its assembly, some of the degrees of freedom have been reduced (~ 10%). The Jacobi preconditioner has a minimal impact on the number of iterations and its computation requires only a few milliseconds. The PETsC library supports the use of additional preconditioner libraries.

With the options *rtol*, *atol*, *divtol*, *max_it* users can control the convergence test of the interactive methods. The convergence of CG is detected at iteration k if

$\|r_k\|_2 < \max(rtol \times \|b\|_2, atol);\ r_k = b - A\bar{x}$ - Residual

or

$\|r_k\|_2 > divtol \times \|b\|_2$

In our tests, we performed 1000 iterations. This number is sufficient to analyse the behaviour of the method without excessive computational time.

One of the major challenges was to initialize the external solver with the test data. The transfer of a linear system (matrix A, vector b) of the application to the solver and the solution of this linear system to the application can be implemented in different ways:

- Linking of the external library (function call).
- Through MPI Communication between two MPI_COMM_WORLDs.
- Through an I/O server architecture by using a file system (e.g. Lustre).
- Through an I/O server architecture by using IO Forwarding.

The first and second methods are undoubtedly the fastest. However they often require more than a trivial additional function call in the application code. The compatibility of the application and the library, as well as load balancing can also be problematic. The use of a file system (in the HPC area, this is Lustre) is the simplest way for communication between an external application and the solver. However, this method is slow. An alternative to communication via the file system, is provided by an I/O server architecture that uses IO Forwarding. Some nodes in the system can be used as the buffer nodes. These nodes store the files with the system of linear equations in memory. The application exchanges data with external solvers through these buffer nodes. The communication happens over the high-speed network achieving high bandwidth and good scaling by using several buffer nodes. The main feature of the communication via IO Forwarding is that the programmer can use the general IO interface. We have programmed a system in which the matrix and the vectors are loaded in parallel, distributed and prepared for the solver. Until the IO Forwarding system works, the data must be saved on a storage system. Despite the capable parallel file systems that we have, the bandwidth achieved very modest values (3-4 Gigabyte per second). It shows the importance of an IO Forwarding system. Its performance increases with the number of used channels. Figure 6 and Figure 7 illustrate the distribution scheme on a coarse-grained level scheme. On Figure 6, those processes are combined in groups, which addressing the separate data streams. The red boxes are the processes that read the data (for example, the matrix) and then redistribute it to the processes (which are highlighted with a green colour). In further tests on the Cray XE6 system, we used 10 master processes. If the data is distributed, PETSc (or another solver) can solve the linear system using the green highlighted processors (see Figure 7).

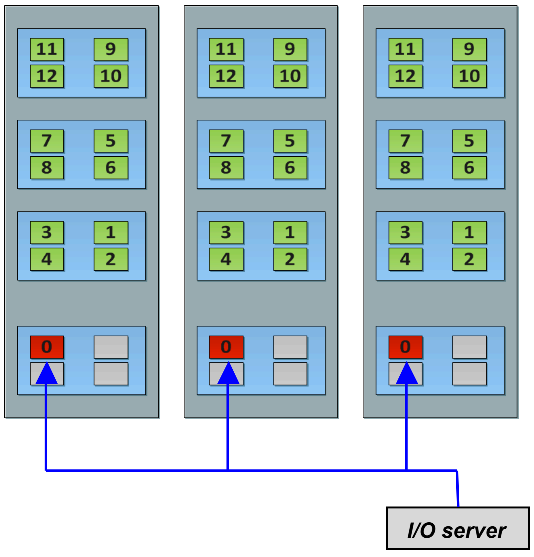


Figure 6 - Distribution of the system of linear equations, part 1

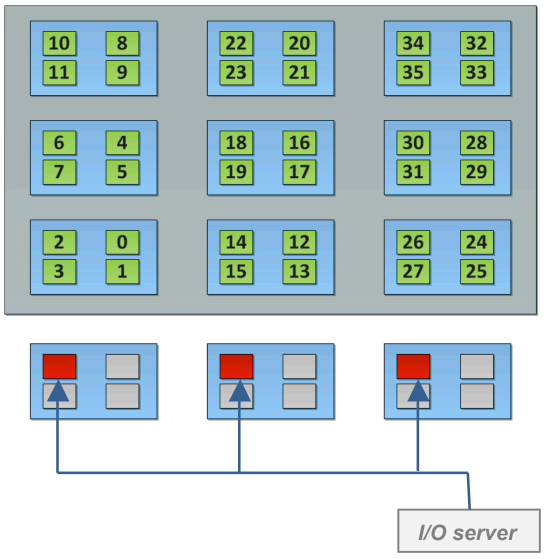


Figure 7 - Distribution of the system of linear equations, part 2

The PETsC library operates with the matrices in distributed (Block-) Compressed Row Storage (CRS) format.

For our tests we have used the symmetric positive definite matrix of the size $\sim 10^8 \times 10^8$ and with $\sim 7^9$ columns. The detailed description of the bone matrix is available in section 8 (Additionally, one other matrix is presented in this section, which we will use in future tests and development). Since the matrix is in the above format, you can easily initialize the PETsC library with it.

All computations were performed on a petaflop system CrayXE6. The technical data of the system is listed in section 7.1.1.

5.1.2 Statistics of the PETsC Library

5.1.2.1 Single and double precision floating-point data types

As mentioned above, we only compute the first one thousand iterations. To achieve sufficient accuracy usually requires many more iterations. In order to solve the system of the bone matrix with the accuracy of 10^{-12} (needed for the bone simulation), we have to iterate approximately ten thousand times. The dependencies of the residual on the number of CG iteration with double and single precision computation are shown in Figure 8. Furthermore, the residuals of the computation with smaller matrix of the same simulation problem are shown (the curves: residual single / double small matrix). This matrix is thousand times smaller (100'000 diagonal elements, with 6’901’344 none zero elements). As noted above, the method converges more slowly as the domain of the simulation becomes larger. After 15’600 iterations with double precision the residual does not decrease and there are no benefits to iterating any longer. After 2’800 iterations with single precision the residual achieves the minimal value of 8.28e-07. The error will not decrease from this. By increasing the size of the problem, the situation will only worsen. But in this case, collective operations, which probably play a primary role in the precision of this algorithm, can be reprogrammed to increase their accuracy. Increasing the accuracy of the collective reduction operations will be considered by the future WP4 task: “Exascale collective reduction collective approaches”). In addition, we observed changes in the precision of the calculation depending on the number of the processes. The first residual double value, calculated on 320 and 5760 processors, differs by 0.001 percent. This difference seems small, but the use of hundreds of thousands (or millions) of processes can lead to a substantial increase in these kinds of errors.

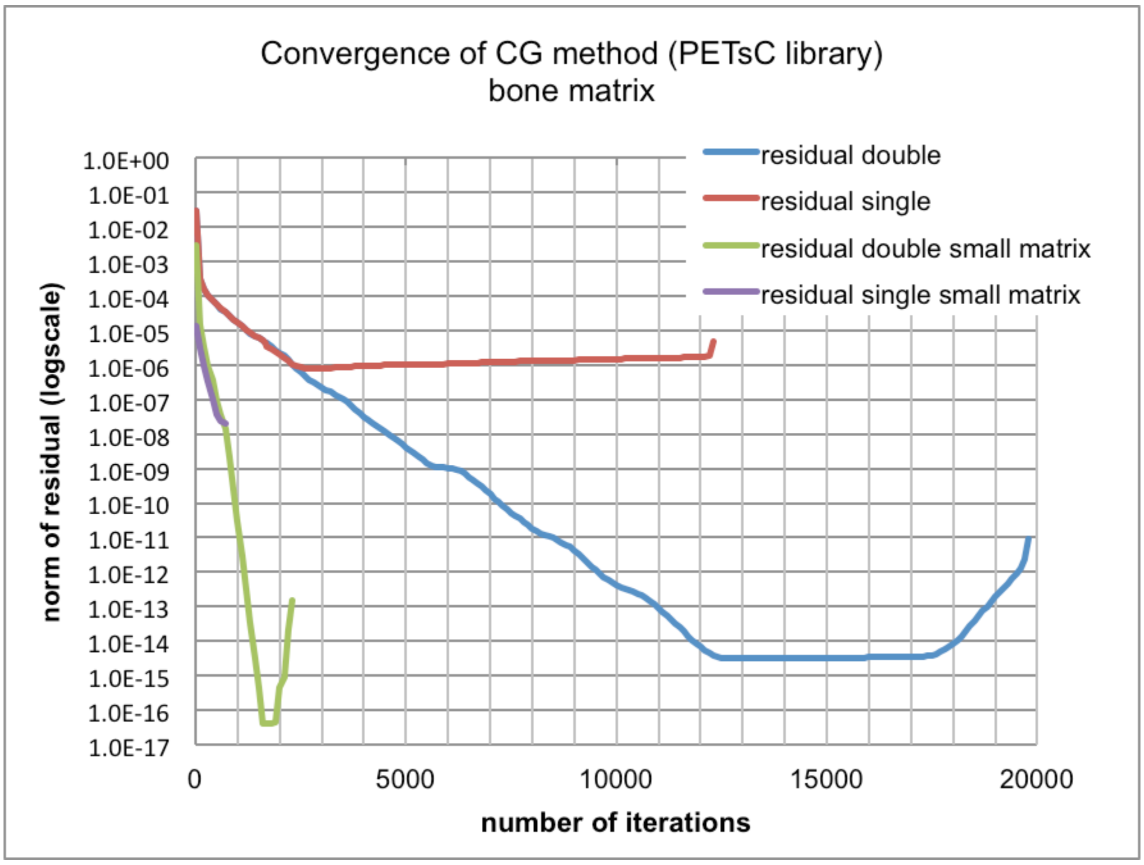


Figure 8 - Dependency of the residual on the number of CG iteration (double, single, bone matrix)

5.1.2.2 Computation with double precision floating-point data type

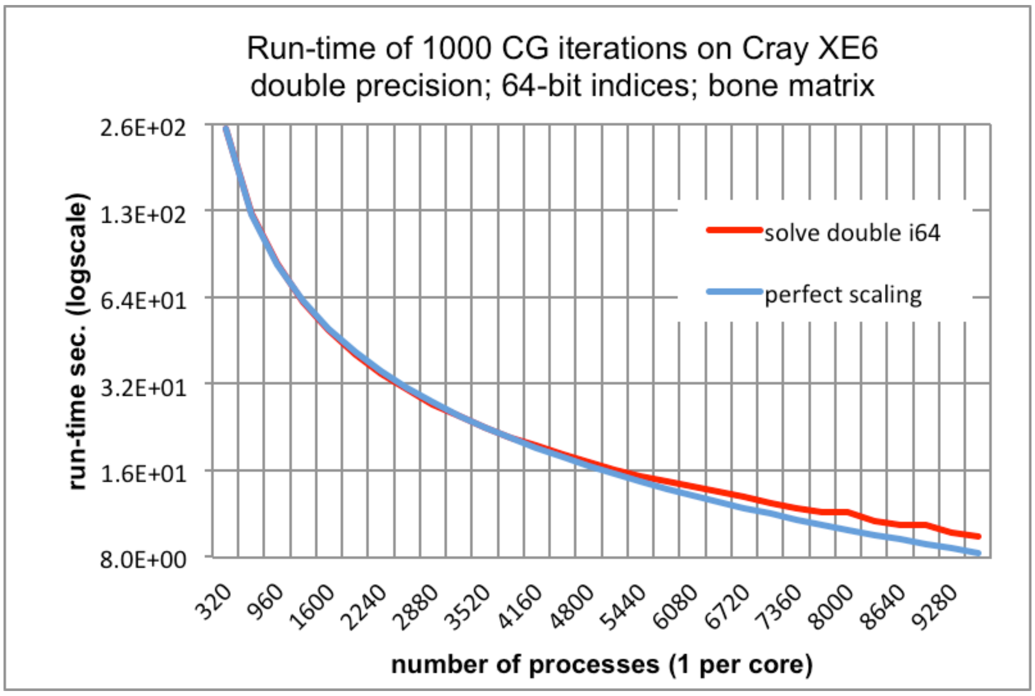


Figure 9 - Run-time of 1000 iterations of CG method on Cray XE6 (double, i64, bone matrix)

Figure 9 shows the runtime per thousand iterations on the CrayXE6 system at HLRS. The computation was done with double precision and a 64-bit integer representation for the indices. To collect the statistical data, the benchmark was run several times. This test was repeated several times. The differences in the run time were no more than 2 percent. The graph shows two curves, one for the run-time, while another shows the run-time in the case of ideal scaling. As a starting point for the ideal scaling calculation, the program's run-time on 320 processes has been chosen. The average performance of one process is pictured in Figure 10. The achieved performance is no more than one and a half percent of the theoretical peak performance. This can be explained because the matrix does not consist of the blocks and the CRS format is not optimal for the matrix vector multiplication.

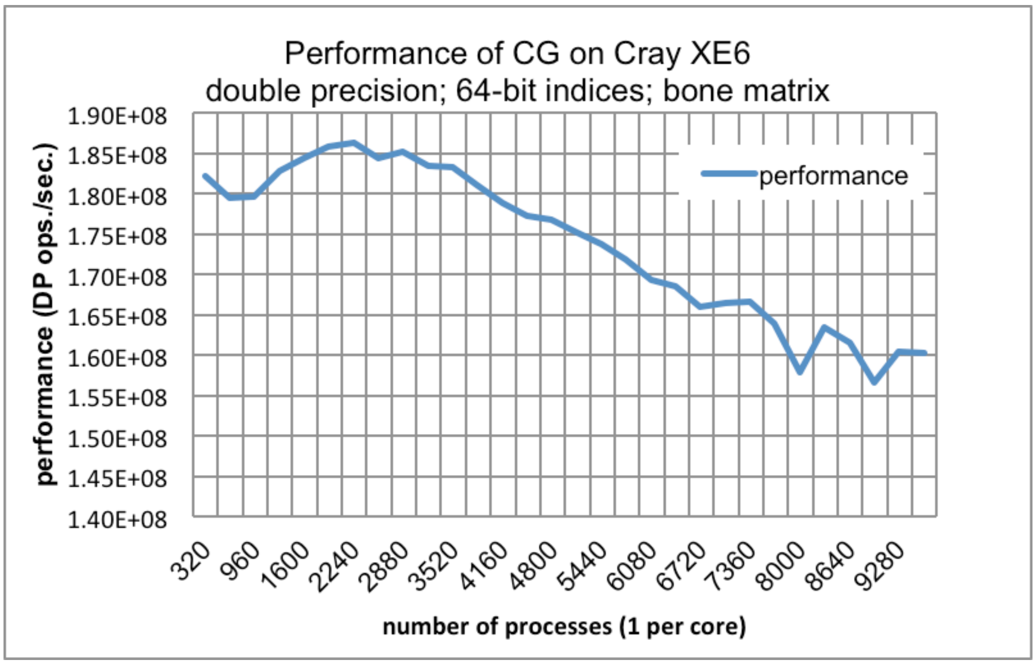
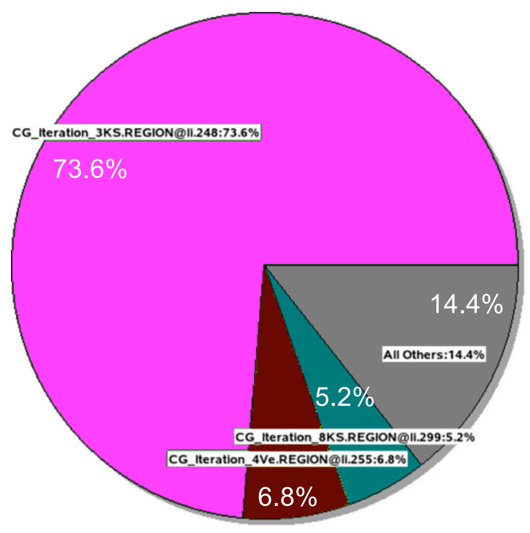
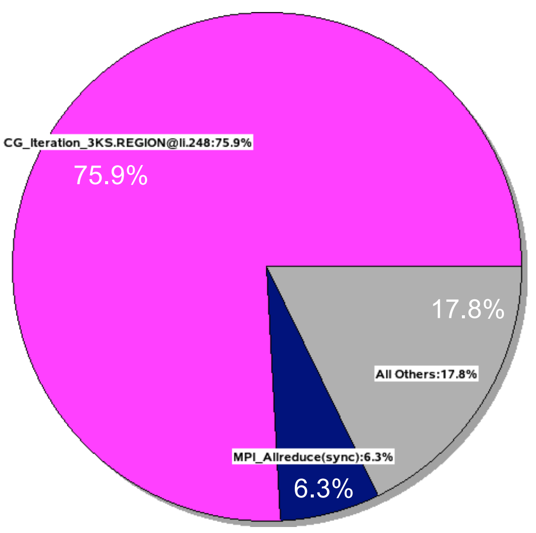


Figure 10 - Average performance of one process during CG iterations on Cray XE6

Matrix vector multiplication (MVM) and the vector operations make up the largest part of the computation. In this configuration, for the operation MVM used asynchronous data transfer with the commands MPI_Isend and MPI_Irecv. To synchronize it the MPI operation MPI_Waitany is used. To calculate the scalar of the vector operations MPI_Allreduce is applied. If we compute on only 320 processes (10 nodes) the MPI ratio of the runtime is low. The most computational parts are shown in Figure 11.





🟪 Matrix vector product

🟥 Vector dot product

⬜ All Others

🟦 MPI_Allreduce (sync)

🟦 Precondition (diag. matrix x vector)

🟩 MPI_Waitany

Figure 11 - CrayPat Diagram of the computation on 320 processes (excludes MPI call profiling)

Figure 12 - CrayPat Diagram of the computation on 320 processes (includes MPI call profiling)

Figure 12 shows the MPI calls that were included in the profiling (with the option "–g mpi"). These two charts show that most of the time of the scalar product has been spent in MPI_Allreduce.

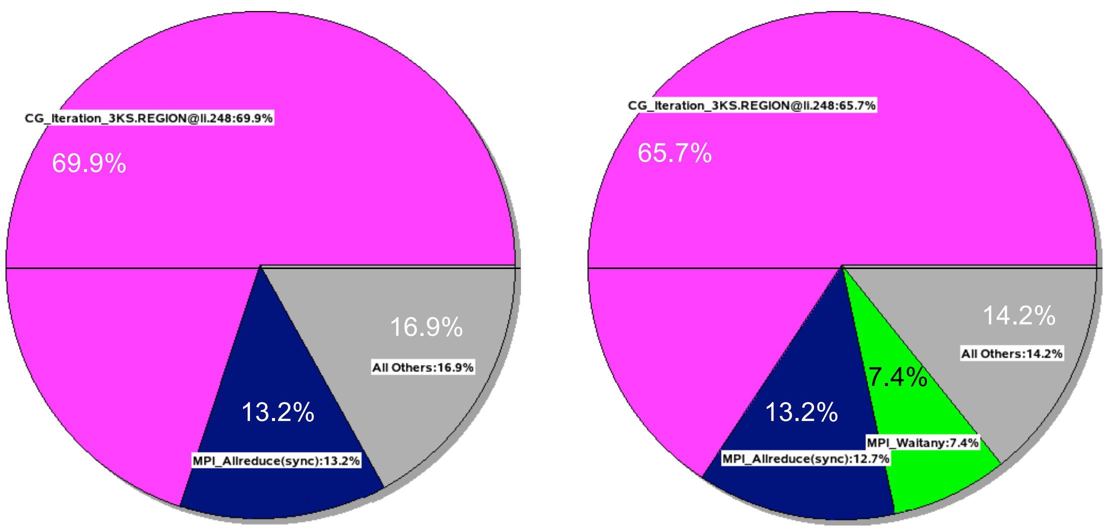


Figure 13 - CrayPat Diagrams of the computation on 4800 and 9600 processes (includes MPI call profiling)

The MPI_Allreduce fraction increases with the number of processes. It amounts to 13.2% of the overall computation with 4800 processes. The MVM communication part increases slowly, but at 9600 processes it begins to become an important factor. For further explanation, we will investigate the distribution of the matrix (load balancing) and the impact of the implementation of MPI collective operations in the CRESTA Co-design team “Linear solvers and pre conditioners”. In addition, we explore the library in the next section. This also gives us an insight into the previously mentioned problems of scalability.

For comparison, Figure 14 shows the execution time of 1000 CG iterations with the small bone matrix. In this case we can also observe even super-linear scaling: data processing takes place in the cache. One node has 32 cores.

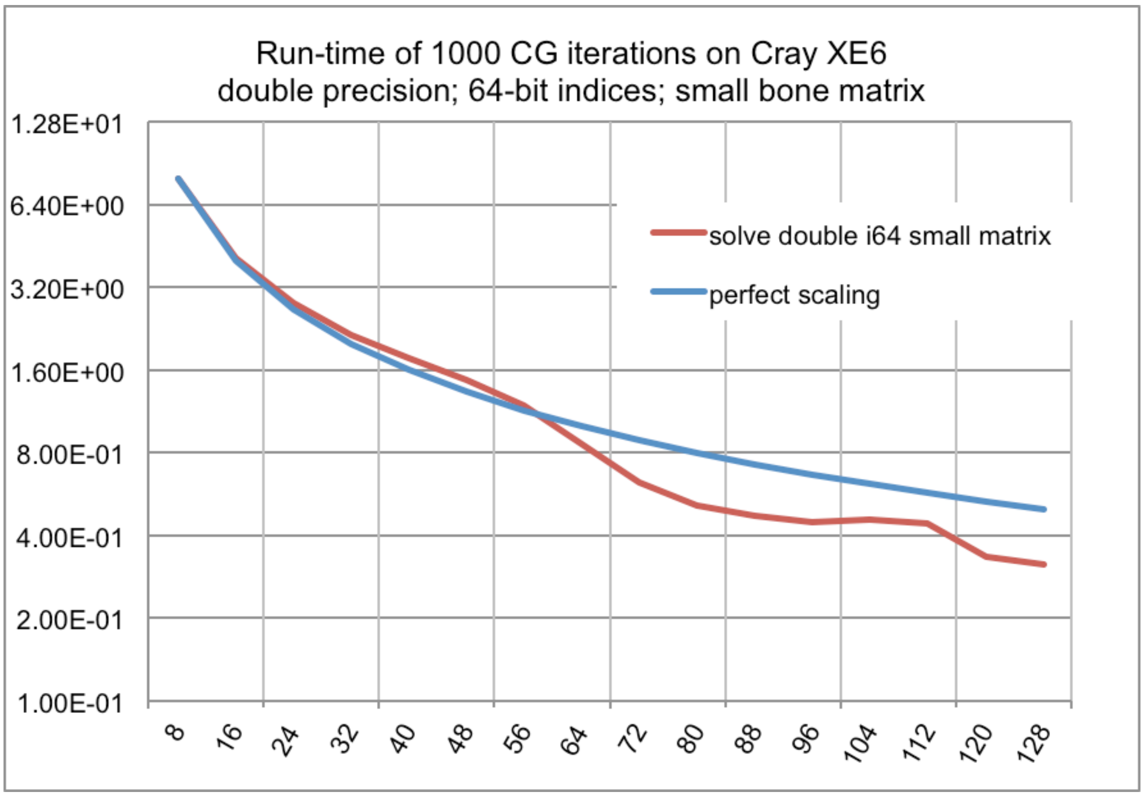


Figure 14 - Run-time of 1000 iterations of the CG method on Cray XE6 (double, i64, small bone matrix)

5.1.2.3 Computation with double / single precision and 64-bit / 32-bit indices

In this section, we carry out a comparative analysis of the computation with different data types. In Section 4.1.2.1 we have shown that the calculation with single precision does not lead to satisfactory results. It is possible, that using more accurate reduction operations will improve the accuracy. In addition, the statistics, collected with different data types, may be of great interest for the evaluation of the prediction model of limiting hardware factors.

Simply using the 32-bit representation of the indices is not always possible. The largest positive number that we can use is $2^{31} - 1 = 2'147'483'647$. The considered matrix is composed of about the seven billion items. The distributed "CRS" format allows us to organize the indexing of its elements, so that we can compute the solution by using more than three processes. However, the $2'147'483'647$ is the upper limit for the number of diagonal elements.

Applying different data types results in undesirable properties:

- The interface between the different modules of the simulation becomes more complex.
- There is a need for copying and reformatting data.
- The probability of programming errors increases.

However the computational time can be greatly reduced. Figure 15 shows the run-time of the computation using different data types for floating-point numbers and indices. The time axis is logarithmic.

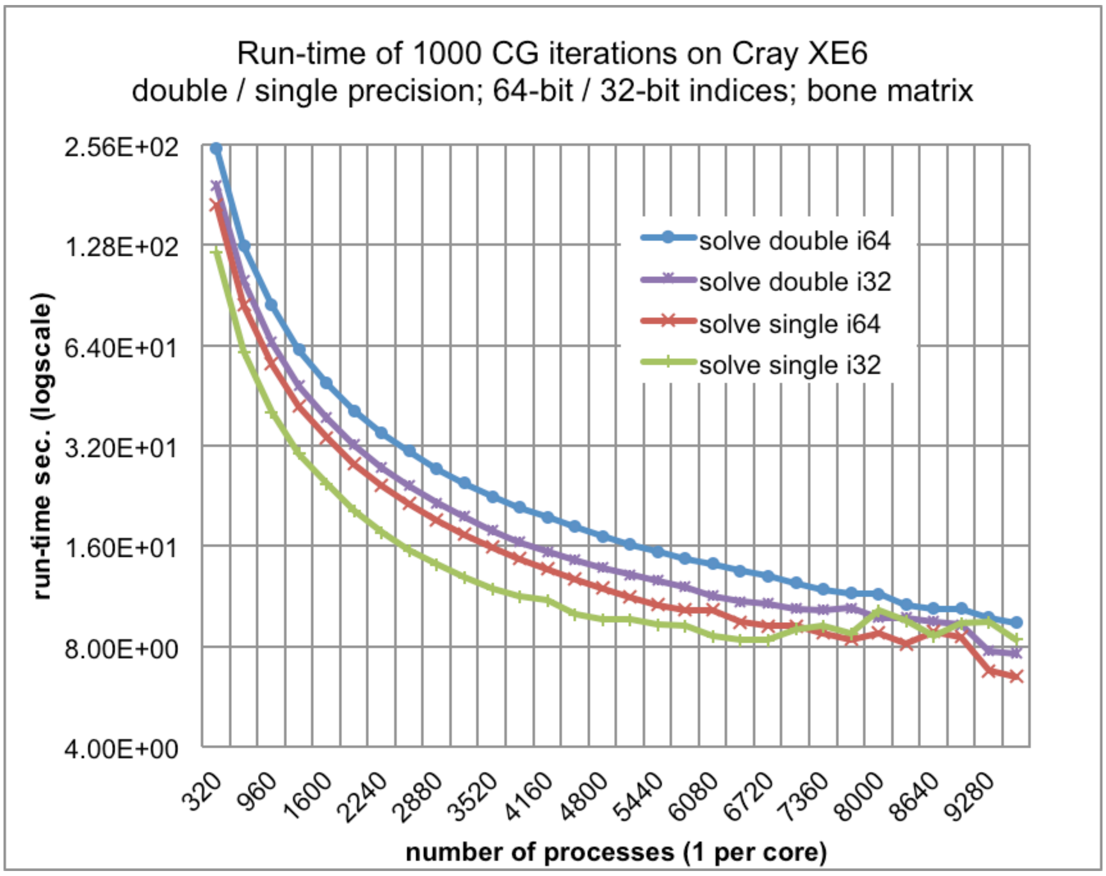


Figure 15 - Run-time of 1000 iterations of the CG method on the Cray XE6 (double / single, i64 / i32, bone matrix)

As you can see, the fraction of operations associated with the floating-point numbers is greater than the fraction of the indices operation. The curve for "single i32" has the

greatest fluctuations, particularly between the points (7680; 8.7674) and (8000; 10.2315). The curve "double i64" has following values at these points: (7680; 11.5821) and (8,000; 11.5364).
The diagrams in Figure 16 show the proportion of parts of computation by single i32 and double i64 on 7680 and 8000 processes.

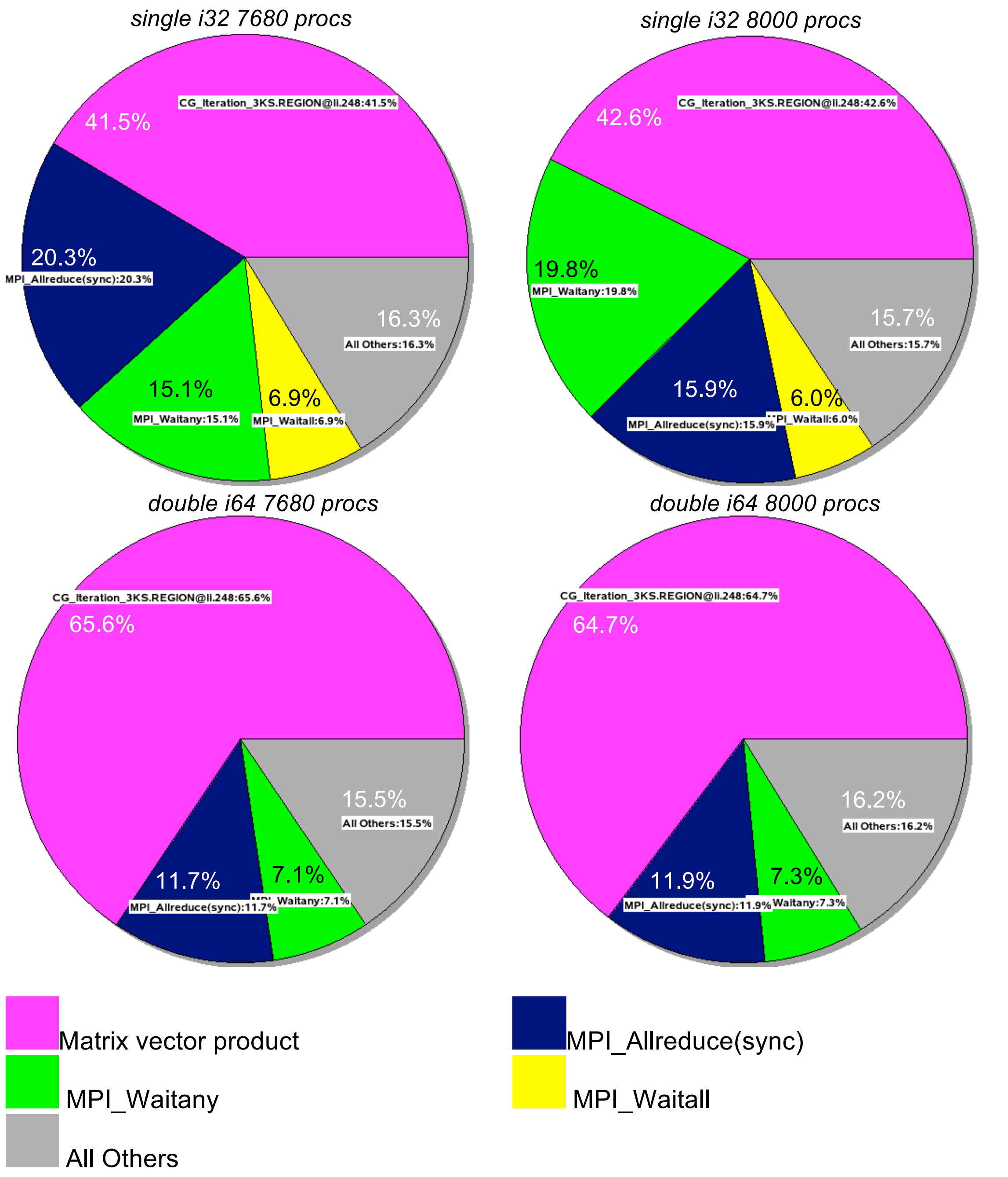


Figure 16 - CrayPat Diagrams of the computation on 7680 and 8000 processes

The operation MPI_Waitany is used in the MVM for the scattering of the vector. The vector data is packed before sending. Hence after the data have been received it can be immediately extracted. It is possible also to use the operation MPI_Alltoallv instead of asynchronous data transfer. To do this, the user has to add a new parameter to the PETsC configurations file (Table 2): *-vecscatter_alltoall*. The run-time increases dramatically if MPI_Alltoall is used: 1000 iterations take around 117 seconds on 7680 processes and 129 seconds on 8000 processes.

As shown above, the use of different data representations can lead to positive results. However, it should also not significantly increase the complexity of the software.

5.2 Algebraic Multigrid

As a representative of the Algebraic Multigrid methods we selected the library BoomerAMG. BoomerAMG is a parallel implementation. It can be used as a solver or as a preconditioner. BoomerAMG is part of the Hypre library (6). It is also possible to directly use the Algebraic Multigrid as a preconditioner in the PETsC library (with linking of Hypre library). As already mentioned, this method is very difficult. It is difficult not only to implement but also to use. Currently, we have not been able to solve the problem with a large matrix with sufficient accuracy (it is still unclear whether it is possible with BoomerAMG). The method diverges after a few iterations. For the small bone matrix the algorithm works without a glitch. The intermediate result is presented in the next Figure. This shows the differences of the residual reduction for the Jacobi and Algebraic Multigrid pre conditioner.

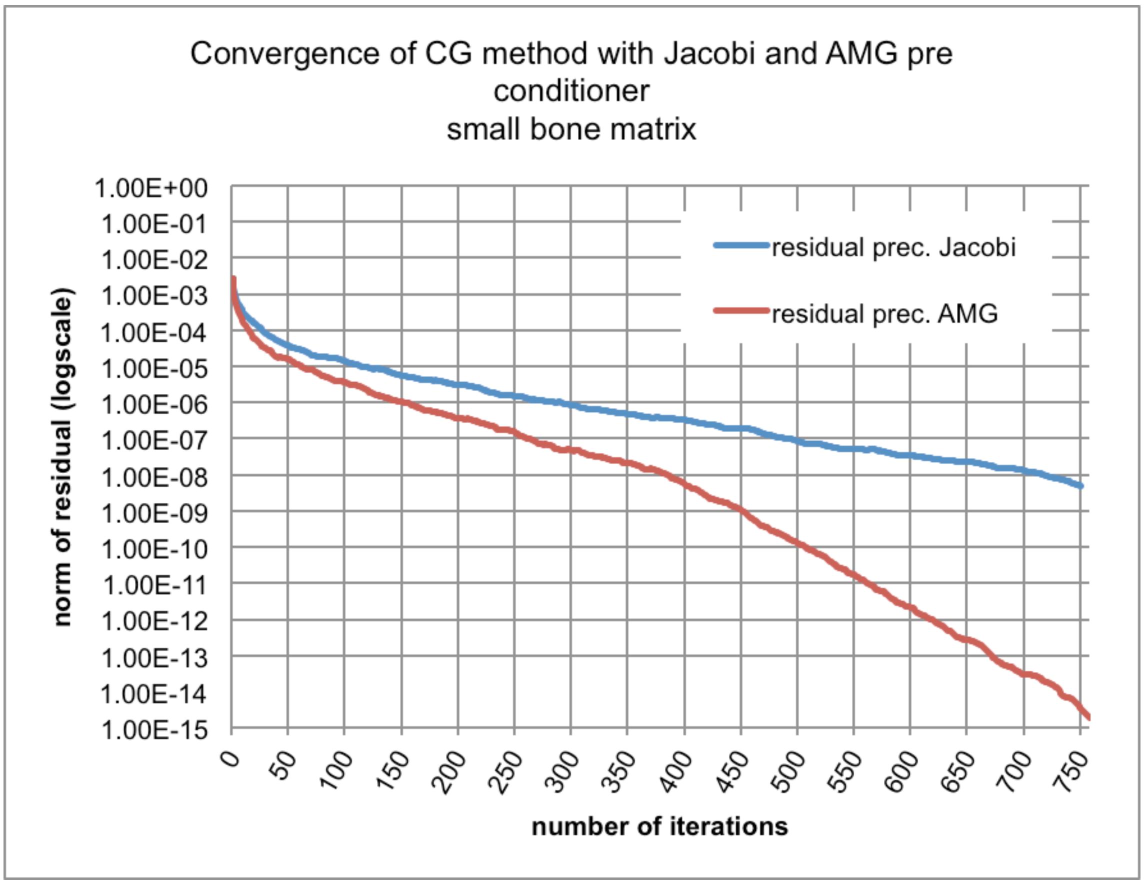


Figure 17 - Convergence of the CG method with the Jacobi and AMG pre conditioner (small bone matrix)

The preparation of the AMG preconditioner (the setup phase) requires a lot of time. In addition, one iteration step takes longer than when using one of the simple preconditioners. So for small matrices it is much faster to use a simple method, e.g. CG with a Jacobi pre conditioner. But when the matrix size is large, we need many more iterations to solve the system of linear equations. Reducing the number of iterations gives enough time for the AMG setup phase. Hence we expect AMG to be relevant for Exascale.

5.3 FFT

5.3.1 Theory

The fundamental limiting factors for FFTs are intrinsic to the algorithm rather than being specific to any one implementation so it is instructive to consider these in the abstract.

As mentioned previously the FFT algorithm has a complexity of O(N log(N)) this can be broken down into log(N) steps each with a potential parallelism of N. The same complexity behaviour applies to multi-dimensional FFTs which is unsurprising because the recursive step used in the derivation of the FFT algorithm is essentially to convert a one dimensional DFT into a two dimensional DFT, which contains the same total number of points; with the additional application of multiplicative phase factors. On modern computer architectures the performance of the FFT algorithm is completely dominated by data movement costs. This data movement is very expensive but is necessary because every word in the output of a DFT calculation depends on every word of the input. This also means it is highly unlikely that any *equivalent* calculation (for example an iterative solver that solves equations in real-space) will require less communication than the log(N) data exchange steps, as required by the FFT algorithm. However for calculations where the full results of an FFT calculation have more information than is required for the solution of the problem then alternative approaches with less demanding communication requirements might be possible. Such examples occur in situations where high frequency components are not required.

In practice, modern machine architectures present a hierarchy of storage locations with different data movement costs. Data movement is much cheaper within caches close to the processor than within lower levels of the memory hierarchy. Similarly with distributed memory systems, communication between nodes is significantly more costly than communications within a node.

The data movement costs of the FFT algorithm can be minimized by arranging the intermediate data layouts to perform as many of the data movements as possible in the higher levels of the storage hierarchy. Similarly with distributed memory systems implementations arrange for as much of the data movements as possible to occur within a node.

In practice most codes that perform FFTs perform transformations on large multi-dimensional datasets (frequently the transform is only applied to a sub-set of the dataset dimensions). In this case it is convenient to implement the overall transform as a series of data redistributions between different data decompositions with each of the active dimensions in turn being local to a node (transpose operations). Between each of the redistributions the local active dimension is transformed using a non-distributed FFT library. The convenience of this approach is because it allows distributed FFT implementations to be built out of optimized single node FFT libraries and highly optimized MPI collectives. However in common with all implementations the overall performance is largely limited by the data movements.

Many highly optimized node local FFT libraries exist. For example the FFTW library: http://www.fftw.org. This library is an example of an auto-tuning library capable of automatically tuning its algorithms at run-time to find a good solution for the specific problem and hardware/software environment encountered by the application. However for distributed memory applications the majority of performance of the inter-node data communications is far more significant for performance than the performance of the underlying node-local FFT library so the choice of which underlying FFT library is largely irrelevant and the performance of the MPI collectives dominate the performance. Though multi node FFT libraries do exist, most applications do not use these, instead each application currently builds their own multi-node FFTs out of a combination of node-local FFT libraries and MPI collectives. Most of the multi-node FFT libraries only support a limited range of input and output data decompositions that typically don't correspond to the data decompositions required by real application. They also typically have no performance advantage over what is generally obtained by application specific implementations built out of the same underlying libraries. Most applications use the collective call MPI_Alltoall because in this collective the global communication pattern is known by all participating processors allowing greater scope for optimisation. The MPI_Alltoallv collective allows greater flexibility in data decomposition and may be used by applications where load imbalance considerations are more important than the absolute performance of the FFT.

5.3.2 Statistics

The following figure shows the time to complete different sizes of 3 dimensional FFT on different core counts of a Cray XE6. Three different sizes of FFT are shown 128^3 256^3 and 512^3

In each case 32 MPI tasks per node and a 2 dimensional processor grid was used. Each data-point represents the best performing data decomposition for that problem size and processor count. All communications used the MPI_Alltoall collective operation within one row or column of the processor grid.

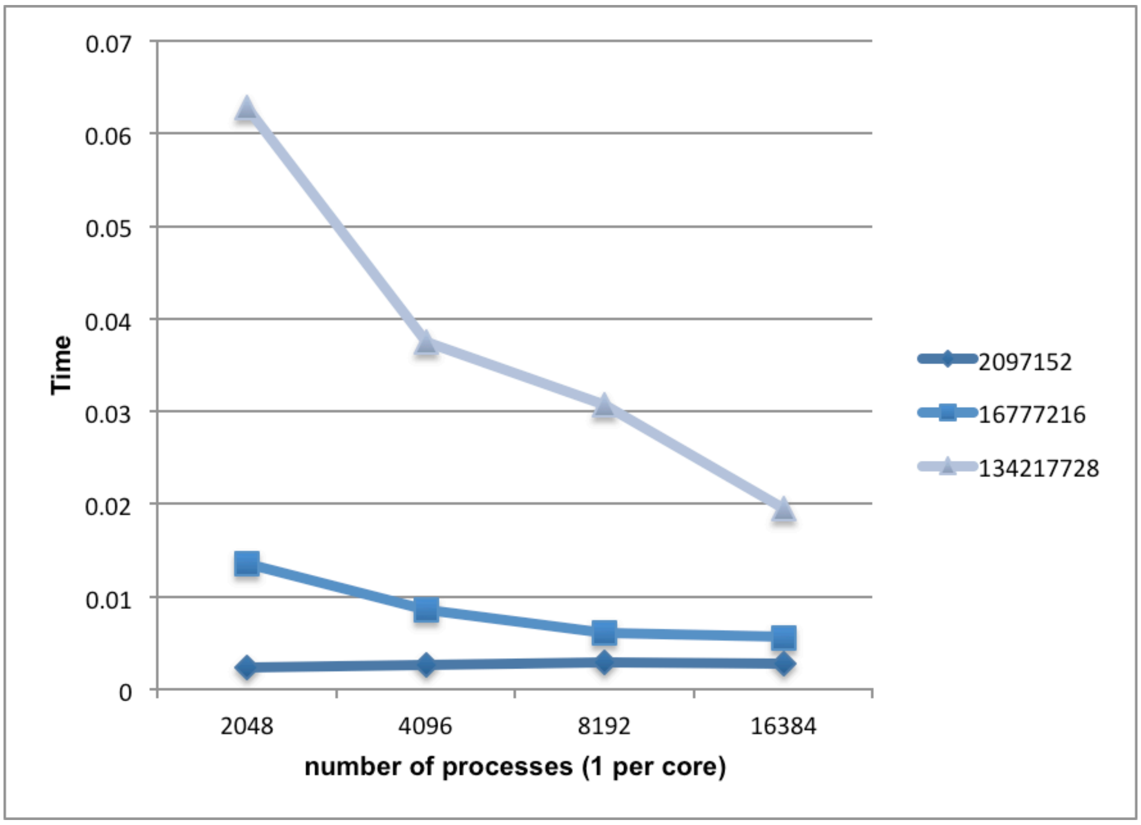


Figure 18 - Performance of 3D FFTs on a Cray XE6

This data shows a clear difference in behaviour depending on how the size of the problem relates to the number of processors. For very large problem sizes the message sizes within the MPI_Alltoall are of a reasonable size and the communication time will be limited by the available bandwidth (either within the node network interface or the bisection bandwidth of the network itself) the aggregate bandwidth increases as node count increases so for large problems the run-time decreases with the number of nodes. For small problem sizes the message sizes within the MPI_AlltoAll are small and communication time is limited by per-message costs such as message latencies. It seems reasonable to consider the small problem example as indicative of performance extrapolated to Exascale.

5.3.3 Model

If we use a modified Latency/Bandwidth model where the message latency may depend on the number of simultaneous messages so the time to send x messages of size S is given by:

$$T_x = l(x) + b.x.S$$

We can expand the latency factor as a power series in x:

$$l(x) \equiv l_0 + l_1.x + l_2.x^2 \cdots$$

- l_0 can be interpreted as a pipelined latency including the network transfer time.
- l_1 can be interpreted as non-pipelined latency for example representing critical sections in the MPI library or contention for the network interface.
- l_2(if significant) could be interpreted as a cost associated with searching internal message queues (each message incurs a cost proportional to the number of outstanding messages). However we will assume terms higher than l_2 are negligible.

We can therefore estimate the time for an MPI_Alltoall (or an MPI_Alltoallv) of a local volume of data V across P processors as being:

$$T_c(V,P) = l(P-1) + b.\frac{P-1}{P}V \approx l_0 + l_1(P-1) + b.\frac{P-1}{P}V$$

If l_1 is large then the performance of MPI_Alltoall with small message sizes can be improved by using a multi-stage algorithm (e.g. (7)) that reduces the overall number of messages at the expense of increasing the overall volume of data sent (some data passes through intermediate nodes before being forwarded to its ultimate destination). This is a logarithmic algorithm where half the data is exchanged with a peer processor at each step giving a communication cost of:

$$T_c(V,P) = \log_2 P.(l_0 + l_1 + b.\frac{V}{2})$$

The MPI library on the XE6 implements this optimisation for small message sizes and performance of the smallest data size above is consistent with a cost logarithmic in the number of processors.

This suggests that the ultimate limiting factor in the performance of the distributed FFT operation is the performance of the MPI_AlltoAll operation (or equivalent) and this is in turn limited by the non-pipelined message latency. The figure shows that in the current technology non-pipelined latency is relatively large and is limiting the scalability of distributed FFTs. Though particularly relevant to distributed FFTs this term is also important for many other communication patterns and will have to be reduced in order to produce usable Exascale systems. There are fundamental limits on message latency. One of these is due to the speed of light though this applies to l_0 rather than l_1.A limiting value of l_1 can be derived from the size of the message protocol header multiplied by *b*.

In principle it is possible to use single sided communications to overlap some of the data movement with the calculation of the local FFTs however this requires the use of many small messages and would therefore also be very sensitive to communication latency.

5.4 Trilinos

First we give a general overview or Trilinos followed by a short description of some packages selected regarding the probable needs of CRESTA in general and WP4 more specifically. How Trilinos can match these is explored afterwards.

5.4.1 Overview

"The Trilinos Project is an effort to facilitate the design, development, integration and on-going support of mathematical software libraries within an object-oriented framework for the solution of large-scale, complex multi-physics engineering and scientific problems."(8)

Based on a package system Trilinos facilitates interoperability. Every package is a self-contained collection of code dedicated to one specific subtask like a direct solver implementation or multilevel preconditioner. It is mostly maintained by a small group with expertise on this particular subject and can therefore match the state-of-the-art. Licensed under LGPL or BSD Trilinos is open source software and thus open for participation. Given implementations can be analysed closely and if they don't meet CRESTA requirements the available packages can be enhanced as needed.

5.4.2 Short package description

The basis of Trilinos is provided by the package Epetra. It provides a uniform interface to various objects like matrices and vectors and thus represents a common language that every package has to be capable with. Furthermore, it can be used to access the well optimized BLAS and LAPACK routines.

AztecOO contains several iterative algorithms that can be applied optionally as a pure solver, a preconditioner for the former or in multiple combinations for both. Although using a somewhat different approach the Belos package yields the same functionality. In this context the package ML should be mentioned which is furnished as a Multigrid preconditioner, of course with a vast amount of possible settings.

To get a decent load balancing Zoltan and its Epetra-interface Isorropia is at hand providing different graph partioning methods. It also implements interfaces to PTScotch and ParMETIS likewise and is therefore particularly useful to run those three against each other in a common environment.

Apart from this Trilinos supplies various tools internally. A command line parser is present for run-time adjusting own programs, Flop counts can be performed in different manners, error handling is widely supported.

5.4.3 Trilinos within CRESTA

5.4.3.1 WP4: algorithms and libraries

Once a matrix interface for application data is available it can be used to set up the corresponding Epetra matrix. From this point on Trilinos supplies various possibilities to precondition and solve this matrix with only a few steps. Thus it is quite easy to get an overview over different algorithm-application interaction under even more different circumstances. Using this approach an exhaustive investigation of the algorithms provided by Trilinos is achievable. Monitoring precisely the outcome hints for an improvement of the algorithm should be possible. If the supplied parameter list for the application specifically best algorithm isn't sufficient enough or a completely new method is needed, a new Trilinos package could be written covering these issues.

5.4.4 Other WPs

Another advantage of Trilinos is its interoperability that is also needed internally in CRESTA. Especially regarding the pre- and post-processing (WP5) the algorithms (and their footing) aren't separated at all. Hence building both (or even more) work packages on the same foundation could lead to overall benefits.

5.5 Collective Operations

It almost goes without saying that the collective communication phases of a parallel application will be the most likely algorithmic components to limit scalability. In the benchmark study of the co-design applications (D2.6.1) collective operations were identified as a potential scaling liability for Nek5000, HemeLB and IFS.

We have shown that for codes using linear solvers the residual calculation can be a limitation on scalability with the MPI_Allreduce reduction operation taking a significant fraction of execution time.

More generally we expect that any code using MPI_Alltoall is likely to suffer challenges in scalability and have already noted the importance of the collective communication phase in FFT implementations in the previous subsections.

For the IFS co-design application we have undertaken some detailed scalability analysis and this code currently requires MPI_Alltoallv that consumes an increasing fraction of time as the core count or model complexity increases. The non-blocking collectives that will be developed and implemented in later phases of the CRESTA project are likely to help with this issue. One particular approach that is currently under investigation is changing algorithms to use single-sided non-blocking communication that we expect will give opportunities for increased scalability.

At the time of writing the initial benchmark suite became available and this will give us the ability to further study the limiting factors in the collective phases of the co-design applications in order to inform our developments.

6 Conclusions and Future Work

We have considered how our use of solvers, FFT and collective libraries can be a limitation for the CRESTA applications as we aim towards exascale.

For linear solvers initial tests have shown that iterative methods must be used only with suitable pre conditioners. The huge number of iterations is not acceptable for CRESTA Co-design applications. Also of great interest is the performance and behaviour of the solver, if we change the precision of floating point numbers and the format for the indices of 64 to 32 bits. Another important task is to expand the test data.

For the example of reading and distributing large data sets, which is described in 5.1.1, we were able to make some additional important conclusions for our future work:

- The transmission of data through an I/O server is promising, and is relatively simple to implement.
- Further development of I/O Forwarding is a necessary condition for the success of our work.

In the next step of verification of existing algorithms, we want to look closely at the libraries BoomerAMG and Trillinos (described above).

FFT operations are almost entirely communication limited and their scalability depends on the scalability of the communication capabilities of the target system rather than their computational capabilities. The high communication requirements of FFTs mean that they are difficult to scale and where a FFT is an over-specification of the problem alternative algorithms may be preferable. However where this is not the case the FFT algorithm remains exceptionally efficient and Exascale implementations need to be addressed. We have developed some simple but informative models of FFT performance that should help in the evaluation of FFT operations at the Exascale.

Collective operations are important and are a potential scaling liability for the co-design application Nek5000, HemeLB and IFS. More generally they are important for FFT operations, linear solvers and when embedded in applications. Our work on optimized reduction approaches and realizing collectives at extreme scale will address some of these limitations.

In this document we have considered how limitations of various libraries may be an inhibiting factor as we progress towards exascale. Our future work on FFTs, linear solvers and collective operations will address these limitations.

7 Platforms

7.1 HLRS-Platforms

7.1.1 CrayXE6

Figure 19 - Cray XE6 (Hermit)

Technical description (installation step 1)	
Peak performance	1.045 PFlops
Cabinets	38 with 96 nodes each
Number of compute nodes	3552
Number of compute cores	per node 2 sockets with 16 cores each: 113 664
Number of service nodes	96
Processor compute nodes	Dual Socket AMD Interlagos @ 2.3GHz 16 cores each
Memory/node	32 GB and 64 GB
Disk capacity	2.7 PB
Node-node interconnect	Cray Gemini
Special nodes	External Access Nodes, Pre- &Postprocessing Nodes, Remote Visualization Nodes
Power consumption	2 MW maximal

7.1.2 NEC Nehalem Cluster

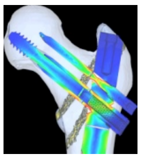
Figure 20 - NEC Nehalem Cluster

Technical description	
Peak Performance	62 TFlops
Number of Nodes	700 Dual Sockel Quad Core
Processor	Intel Xeon (X5560) Nehalem @ 2.8 GHz, 8MB Cache
Memory/node	12 GB
Disk	80 TB shared scratch (lustre)
Node-node interconnect	infiniband, GigE
accelerators	32 nodes provide Nvidia Tesla S1070 GPGPU
graphical pre- and post processing nodes	6 nodes provide NvidiaQuadro 5800FX graphics card

8 Matrix Collection

8.1 Bone matrix

8.1.1 Project



Biomechanical simulations on parallel computers to assist in trauma care and hip endoprosthetics– BiSPar.
Motivation: To improve the patient-specific treatment options and reduce treatment costs.
Developers:

- *HLRS: Ralf Schneider* ;
- LASSO Ingenieurgesellschaft mbH;
- Universitätsklinikum Freiburg;

Figure 21 - Description of the bone matrix project

8.1.2 Short description and geometry

The bone matrix was assembled during the static Finite-Element (FE) simulations of bone-implant-systems. The Figure 22 pictures the small part of the geometrical data of a cortical bone that was used to assembly the matrix. The topology consists of 44’332 hexaeder8 and 58948 nodes.

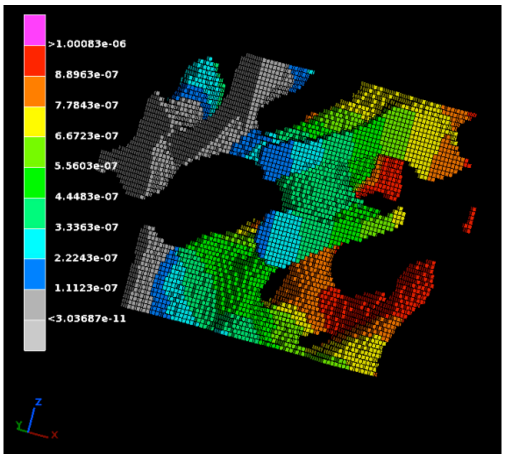


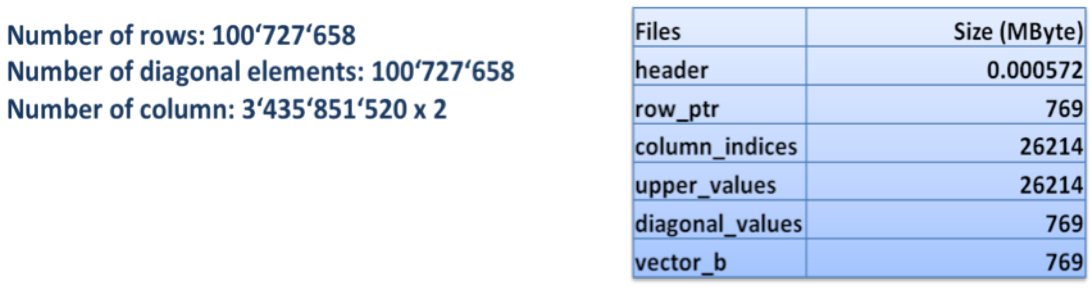
Figure 22 - Model of cortical bone of size 1 cm^{3}with colour figure of displacement.

8.1.3 Properties of the matrix

The bone matrix is symmetric and positive definite. It is strongly diagonally dominant:

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}| \text{ for all i}$$

It is saved in four files in the serial CRS format (binary). Each of the files stores one field. Additional to these files the vector b and the short description of the internal structure are saved in the files (*header* and *vector_b*). The details to the size of the matrix and files are in the following table2.



Number of rows: 100‘727‘658
Number of diagonal elements: 100‘727‘658
Number of column: 3‘435‘851‘520 x 2

Files	Size (MByte)
header	0.000572
row_ptr	769
column_indices	26214
upper_values	26214
diagonal_values	769
vector_b	769

Table 3 - Quantitative properties of the bone matrix

Possible size:

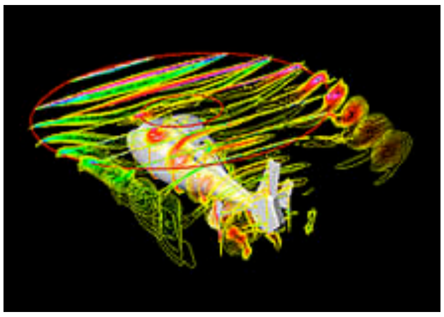
1) *48 x 48* , for interface adjustment / debugging
2) 107’568 x 107’568 (non zeros: 6’793’776)
3) 100 million x 100 million (non zeros: 7.5 billion)

Required size:

A simulation of a relevant part of the knee bone (for treatment) will produce matrices of size $10^9 \times 10^9$ and more.

8.2 Aircraft grids

8.2.1 Project



Prediction of viscous and inviscid flows about complex geometries from the low subsonic to the hypersonic flow regime, employing hybrid unstructured grids.

Developer: *DLR Institute of Aerodynamics and Flow Technology*

Motivation: To improve complex aircraft-type configurations in shape and material design.

Figure 23 - Description of the aircraft grid matrix project

8.2.2 Short description and geometry

Flow Simulation (including coupling to structure and flight mechanics): TAU can be used with both (block-) structured and hybrid unstructured grids composed of hexahedrons, prisms, tetrahedrons and pyramids. The calculations of viscous flow around a delta wing at M=0.5, alpha=9º is pictured on the Figure 24.

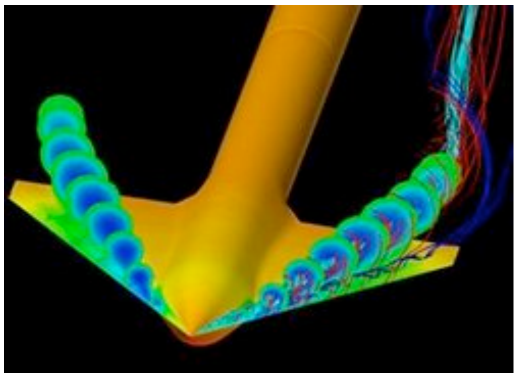


Figure 24 - Calculations of viscous flow around a delta wing

8.2.3 Properties of the matrix

The matrix is non-symmetric, and real. It is stored in binary Blocked CSR Format:

Usual CRS containing a 5x5 block in each matrix position.

Possible size:

1) 4 x 4 2x2-blocks, for interface adjustment / debugging
2) 108‘396 x 108‘396 5x5-blocks, therefore 541‘980 unknowns (non zero:170‘610‘950)
3) on demand (almost) arbitrary

Required size:

The dimension depends on the specific simulation done and the grid size is adjustable and produces better results at a finer setting. Thus 10^7x10^7 could be a first step.

9 References

1. *Algebraic Multigrid (AMG): An Introduction with Applications.* **Stüben, K.** 1999.

2. **Hackbusch, Wolfgang.** Hierarchische Matrizen: Algorithmen und Analysis. s.l. : Springer, ISBN 978-3-642-00221-2, 2009.

3. **Küster, Uwe.** HLRS and Institute of High Performance Computing (Universität Stuttgart). *Modeling, simulation and optimization methods.* [Online] 2011-2012. http://www.ihr.uni-stuttgart.de/lehre/vorlesungen/modellierung-simulation-und-optimierungsverfahren-i/.

4. **Ronald Kriemann.** HLIBpro. [Online] Max-Planck-Institut für Mathematik in den Naturwissenschaften, 2012. http://www.hlibpro.com/.

5. Portable, Extensible Toolkit for Scientific Computation. [Online] 09 08, 2011. http://www.mcs.anl.gov/petsc/.

6. http://acts.nersc.gov/hypre/. [Online] Center for Applied Scientific Computing (CASC) at Lawrence Livermore National Laboratory.

7. *IEEE TPDS.* **al, Jehoshua Bruck et.** 1997.

8. *An overview of the Trilinos project.* **Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle ...** s.l. : ACM Press, 2005.