D4.3.1 – Initial prototype of exascale algorithms and solvers for project internal validation

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:	M24
Submission date:	30/09/2013
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	HLRS
Version:	1.0
Status	Final
Author(s):	Dmitry Khabi (HLRS), Stephen P Booth (UEDIN)
Reviewer(s)	Lorna Smith (EPCC), George Mozdzynski (ECMWF)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	18/08/2013	First draft for comments	Dmitry Khabi (HLRS)
0.2	30/08/2013	FFT part	Stephen Booth (UEDIN)
0.3	07/09/2013	Linear-Solver Part	Dmitry Khabi (HLRS)
0.4	09/09/2013	Final draft version	Dmitry Khabi (HLRS)
0.5	16/09/2013	Updated due to the internal review	Dmitry Khabi (HLRS)
0.6	17/09/2013	Updated due to the internal review	Stephen Booth (UEDIN)
1.0	19/09/2013	Final version	Dmitry Khabi (HLRS)

Table of Contents

Index of Figures

Unknown Field Co
Unknown Field Co
Unknown Field Co
Unknown Field Co

1 Executive Summary

Deliverable D4.3.1 is a software deliverable. This document describes the software, a prototype parallel numerical library targeted at Exascale systems. The software is available on the CRESTA SVN server at:

https://svn.ecdf.ed.ac.uk/repo/ph/cresta/wp4/cresta_libraries/cel

As previously discussed in the deliverables of WP4 "D4.1.1 *Overview of major limiting factors of existing algorithms and libraries"*(1)and "D4.2.1 *Prediction Model for identifying limiting Hardware Factors"* (2) the Exascale is going to require an increase in the efficiency, in the sense of scalability and performance, of algorithms due to the very large degree of parallelism that will be required. As well as efficient algorithms highly efficient implementations of those algorithms are also required. In addition to the increase in the degree of parallelism Exascale systems are expected to be significantly more complex than current systems with many different levels of memory and communication hierarchies. This will make it very difficult to optimize codes for Exascale systems. Many codes will require significant rewriting to make the best use of these systems. The availability of parallel numerical libraries designed for Exascale systems should significantly reduce the development costs of this process. We have evaluated a number of existing numerical libraries that implement linear solvers (such as PETSc (3) or Trilinos (4)) though these are scalable on current hardware they haven't achieved, in our opinion, the highest possible efficiency (see more details in (2) and (1)). In addition current solver libraries do not properly address key issues at the Exascale such as the overlap of communication and calculation. Though Fourier transforms are an important part of many simulations and node-local FFT libraries are widely used, most major applications implement their own distributed FFTs using a combination node-local FFT libraries and explicit MPI communications. We believe that this is because the currently available parallel FFT libraries place too many constraints on the data decomposition of the rest of the application.

For all of the above reasons we are developing a new library (the CRESTA Exascale Library, CEL in short) addressing these two important classes of numerical problem: linear solvers and multi-dimensional Fourier transforms. This initial prototype of the library will form the basis for further testing and improvements. Ultimately the optimized library will be integration with the CRESTA applications:

The Spectral Transformations used within the GROMACS application requires complex collective communication to perform changes in the data decomposition. This kind of communication seems to be one of the weak links for exascale computing. Therefore, our approach emphasizes the communication component of the FFT problem.

One of the challenging areas for exascale computing in the applications OpenFOAM and ELMFIRE is the use of linear solvers. It's necessary to use a fast scalable linear solver especially in the simulation of an entire hydraulic machine using Large Eddy Simulation (see (5) for more details). In this deliverable we describe a model for the distribution of the matrix and vectors, which allows carrying out the matrix vector multiplication with an emphasis on overlapping of computation and communication.

The CEL also provides a framework for the development and evaluation of some of the other new and promising disruptive technologies being developed within the CRESTA project for example:

- WP2: The power measurement across algorithms

  It should be possible not only to measure the power consumption of the CEL library algorithms but also to optimize it for efficient power consumption: for example by adjusting of the CPU frequency during execution and analyzing the effect on performance and power consumption. The details of our first investigations in this field can be found in (6).

- WP4: CRESTA Collective Communication Library (CCL) (7) and CRESTA Microbenchmark suite (8)

The non-blocking collectives are a new development in HPC. The CEL library uses new approaches for collective communication bymatrix vector multiplication in combination with remote-memory access. Together with the developers of the CCL library we are going to investigate, how it would be possible to integrate the CCL Library into CEL library and what is the best way to organize the overlapping during the multiplication.

- WP4: Optimized Reduction Approach

It's not always clear how far the distributed reduction approach affects the numerical calculations in the case of a real application. It is planned to continue the investigation using the CEL library to collect the necessary statistics. The details relating to the previous investigation of using multi-precision software can be found in (1).

2 Introduction

The deliverable software is called CRESTA Exascale Library. The library consists of two parts: Linear solver and Multi-dimensional FFT. The CEL-FFT library is intended to make it easier to implement FFT implementations in particular to allow the data decompositions used elsewhere in the applications to be less constrained by the needs of the FFT. The CEL-Linear-Solver is used to solve large linear systems. It's planned, that the final version will contain a set of the iterative solvers with Jacobi- and AMG-preconditioner.

The CEL library is being developed using the parallel paradigms Hybrid MPI/OpenMP and Coarray. The initial prototype of the library can be found in the CRESTA SVN. The library is still in its early development state. More alternative implementations, optimizations and features will be added into it.

We have a set of the data from real applications that will be used for the internal validation. The re-implementation of the bespoke FFT code and linking of the Linear-Solver library into the selected CRESTA applications are the further steps in the development and validation process.

The first part of this document contains the description and the development state of the CEL-FFT library. The second partprovides detailson the CEL-Linear-Solver library.The descriptionof the interface, compiling and runtime parameters of the first public version will beprovided in the next deliverable D4.3.2 of work package 4 after further improvement and validation.

3 CRESTA-FFT library

The CRESTA Exascale FFT library is intended to make it easier to implement FFT implementations in particular to allow the data decompositions used elsewhere in the applications to be less constrained by the needs of the FFT. We have been developing a generic library (reshape) to support changes in data decomposition that can then be used to quickly optimize the FFT strategy for the available hardware.

In the next two sections we describe the advantages of this library and how it works.

3.1 FFT algorithms

The parallel performance of the FFT algorithm is best understood by considering the data movement graph for the algorithm. For the FFT this is the "butterfly" pattern (see Figure 1). As can clearly be seen from the graph representation a 2^n FFT has a computational complexity of O(N log(N)). The algorithm has a potential parallelism of O(N) with good load-balance but is also a non-local algorithm requiring a high degree of data movement. The computation performed by each node of the graph is quite small so the time to execute these communication steps will dominate performance at large numbers of nodes.

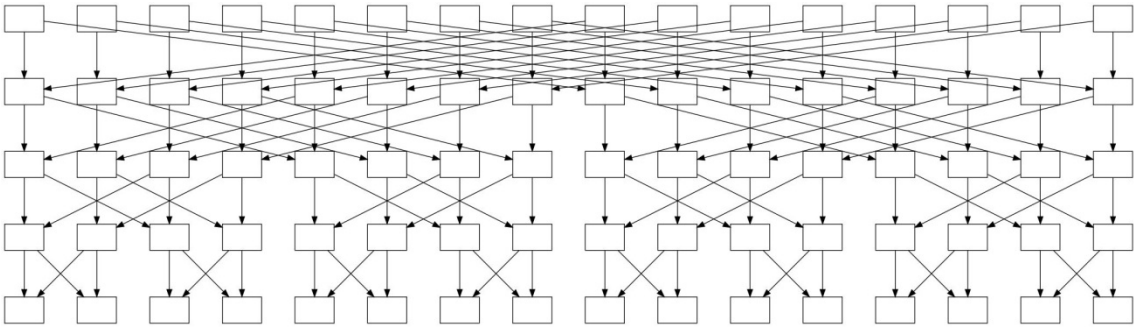


Figure 1 - Data movement graph for a 2^4 FFT

The most common use of Fourier transforms in HPC applications are as multi-dimensional FFTs.

Multi-dimensional FFTs are usually implemented by performing each dimension of the FFT in turn. This gives rise to the same graph representation as a single large FFT consisting of the same number of points. The only difference between the two algorithms being the phase factors applied at each computational stage. In a multi-dimensional FFT there are no constraints on the order that each dimension is processed. In fact stages of the FFT algorithm from different dimensions can be interleaved provided the order within a dimension is preserved. However this only changes the assignment of initial data points to the initial graph nodes. The topological structure of the graph always remains equivalent to a graph for a single FFT of the same size.

An efficient parallel implementation strategy for any FFT calculation therefore depends on choosing a set of data decompositions for each stage of the FFT so as to minimize the amount of data that needs to cross node boundaries. For an Exascale architecture with multiple levels of communication hardware this optimization may take place on many levels. In general an initial communication stage is required to place the data in the correct starting decomposition but in some cases it is possible to adapt the surrounding application to use this same decomposition.

Current parallel multi-dimensional FFTs usually avoid this initial communication phase by starting from data decompositions where at least one of the dimensions are local to a node. Node-local FFT implementations are then applied to each dimension in turn interspersed by communication phases where necessary to change the data decomposition so that the FFTs in the next dimension become local (see Figure 2).

This also has the practical advantage that existing single node FFT libraries can be used to implement the computational phases. Unfortunately a similar approach is not usually possible within a single dimension because the initial stage of the FFT is over the longest length scale and the initial communications step can only be avoided if the rest of the application is able to utilize data in a cyclic decomposition.

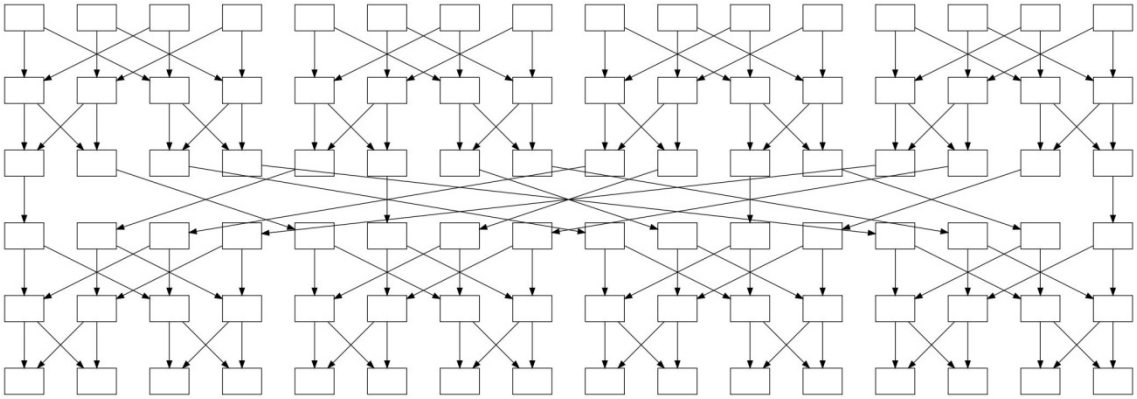


Figure 2 - Graph representation of a (2^2x 2^2) 2D FFT

These transpose based implementations require one less communication stage than the number of dimensions in the FFT (pencil decomposition). At low numbers of nodes this can be reduced by having decompositions where more than one of the dimensions are local at the same time (slab decomposition).

It is also possible to decompose the global data graph in a way that the transformations over some of the dimensions are spread over more than one data decomposition (a split-transform). This gives greater freedom of choice in the number of processors that can be used. Unfortunately it is unlikely that the initial data redistribution phase can be avoided in this case so at low node counts a slab-decomposition with a single communication step will be preferable. At higher node counts (up to the square root of the total number of points in the FFT) this approach will be comparable to the pencil decomposition but will not constrain the data decompositions of the rest of the application. Depending on the balance of the communication and computational abilities of the target hardware at very high node counts it might be more advantageous to run the internal phases of the FFT on a subset of the computational nodes rather than introduce more communication steps.

3.2 Use of CRESTA-FFT library

All of these different strategies for the implementation of distributed FFTs come down to changes in data decomposition. Current practice is to laboriously hand code each strategy within each application making it difficult to explore new strategies as hardware changes. We have been developing a generic library (reshape) to support changes in data decomposition that can then be used to quickly optimize the FFT strategy for the available hardware.

In general any data decomposition can be represented at run-time by an object with the following interface:

- A method that maps global coordinates to a process rank
- A method that maps global coordinates to a local memory offset.

To start with, we are only considering decompositions where each dimension of the data-set is decomposed independently. We can therefore represent the decomposition along each dimension as a separate object and combine them using a set of processor-rank and memory-offset stride values. This is still capable of representing a

far more general set of decompositions than most parallel libraries. The downside is that these general decompositions are a little more expensive to use.

To mitigate this, we use an interface which uses decomposition descriptors to build re-usable communication plans for switching between data decompositions (essentially these are lists of MPI datatypes corresponding to the necessary messages). Any additional overhead only takes place in the initial planning stage and should have little impact on the overall performance of the code. As an added bonus virtually the same code can be used to build MPI-IO file-view datatypes to support parallel IO to the different decompositions.

3.3 Performance comparison of different implementations

To validate this approach our first step was to reproduce the capabilities of the distributed FFT provided by the FFTW-3 library on HECToR. The parallel FFTW library only supports decomposition in one dimension so it is relatively easy to reproduce. The following graph shows the time to solution of two different sizes of 3D FFT. These tests were run with one MPI task per node but with multi-threading enabled within the local FFTs.

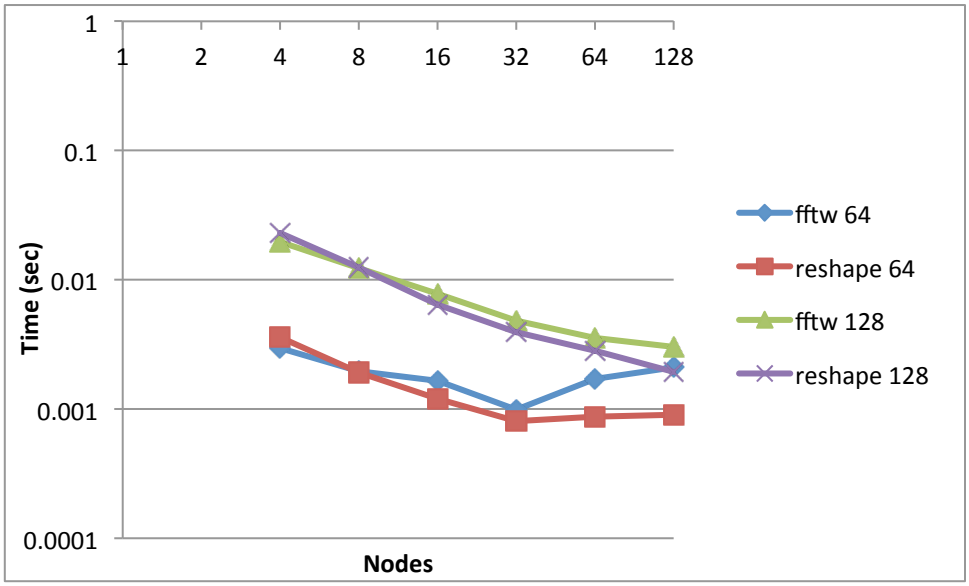


Figure 3 - The time to solution of two different sizes of 3D FFT on HECToR

3.4 Further work

The following work is planned to be performed for improvingthe performance and libraryinterface:

- Compare the performance of a reshape based implementation with a hand coded pencil decomposition.
- Implement and benchmark the new split-transform strategies
- Validate the ease-of-use of the library interface by re-implementing the bespoke FFT code in real applications (e.g. GROMACS) and comparing performance.

4 CRESTA-Linear-Solver library

The CEL-Linear-Solver library runs with at least two threads per process. The master threads are responsible for the communication between processes, allocation of the data structures on the shared memory and some mandatory short calculations. The worker threads perform calculation in parallel using the shared memory. The synchronization between threads occurs through omp barriers and protected variables in the shared memory. The master threads read the data from the remote memory and store it in the shared memory (e.g. by MPI_Get).Figure 4 illustrates the parallel programming model used in the CEL-Linear-Solver library.

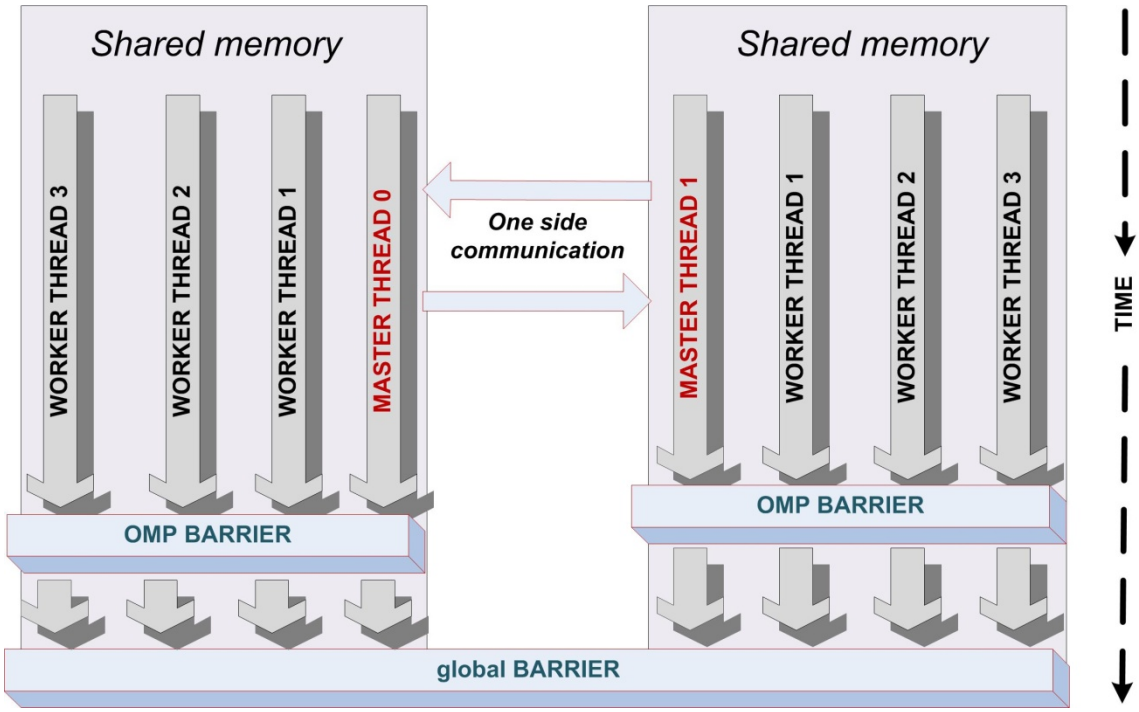
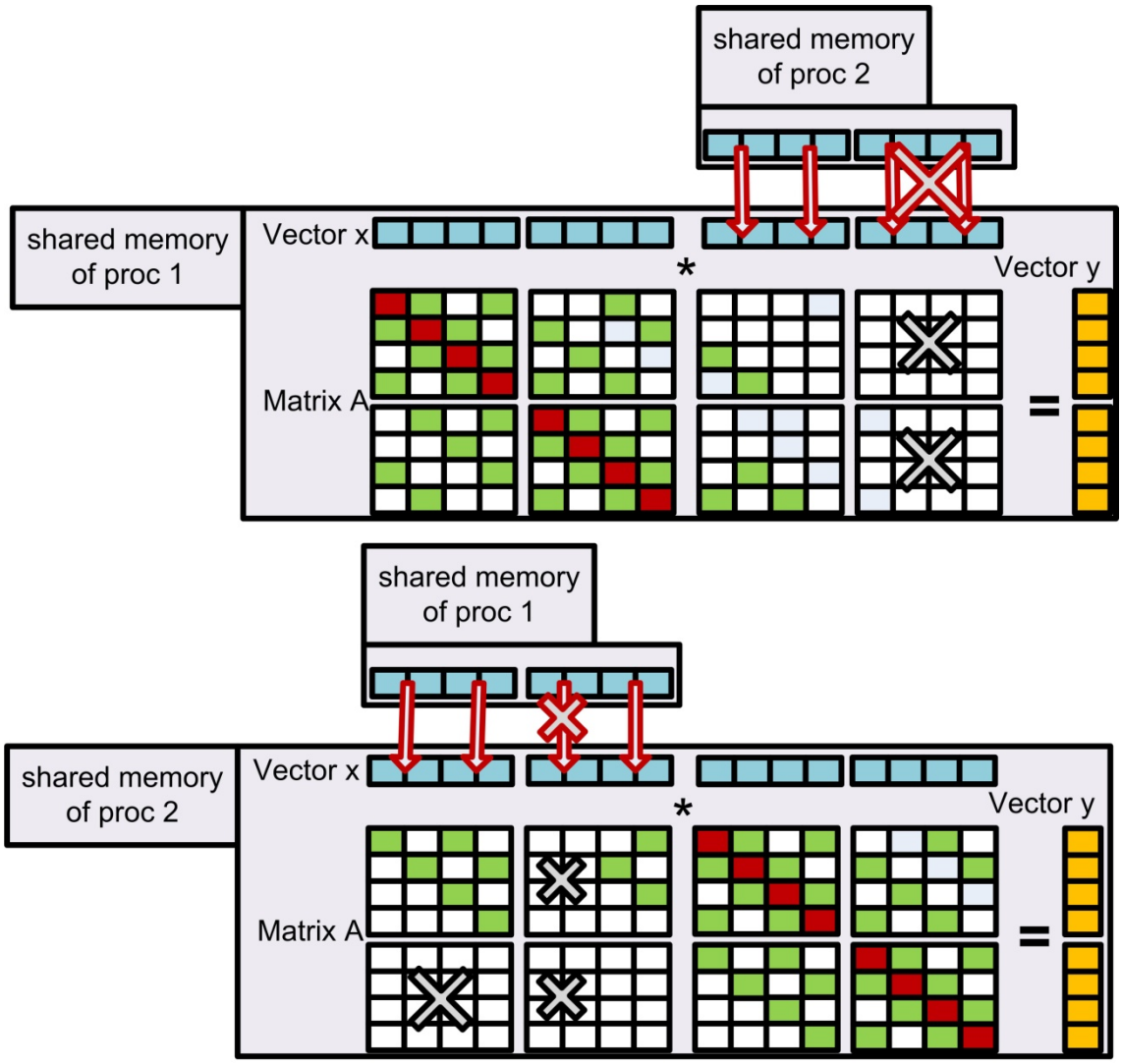


Figure 4 - Parallel programming modelof CEL-Linear-Solver library

4.1 Sparse matrix and vector distribution

The execution of the matrix-vector multiplication (*A*x=y*) is one of the most complex and time consuming operationsfor solving the linear system. Overlapping computation with communication is one of the basic techniques to improve its efficiency. To make it possible we have to distribute the matrix, vectors data and the calculation on it in a particular way that has some important differences to the most widely used methods. The sparse matrix A is divided into the sub-block matrices with number of rows and columns equal to *block_size =num_chunks x chunk_size*. The *chunk_size* is equal to the number of elements of the vector x, which will be read from the remote memory at once (e.g. by MPI_Get(...,*chunk_size*,…)). Figure 5 shows the example for the case of two processes and a sparse matrix of size 16 x 16. The chunk consists of two elements. Two chunks compose one block. Since the matrix is sparse, there are many zero sub-blocks. Depending on the distribution of the non-zero elements in the sparse matrix and the parameters *num_chunks and chunk_size* will be decided what part of the vector *x* that will be read by the master thread from the remote memory. When the transfer operation is completed, the master thread changes the state of the chunk from remote to the local. If all needed chunks of the block are available locally one of the worker threads perform the multiplication of the sub-matrix and vector x.



num_processes = 2; matrix_size = 16 x 16; chunk_size = 2; num_chunks = 2
block_size = chunk_size x num_chunks; sub_matrix_size = block_size x block_size

Figure 5 - Distribution of the sparse matrix *A* and the vectors *x*, *y* by the matrix vector multiplication *A*x=y* implemented in the CEL-Linear-Solver library

Figure illustrates five possible states of data distribution regarding one vector block:

- The block vector is local and no data must be transferred
- The sub matrices are not zero and all chunks must be transferred from the second processor.
- The sub matrices are zero and no data must be transferred
- One of the sub matrices is zero but all chunks must be transferred
- One of the sub blocks is not zero but only half of the block (one chunk) must be transferred for the calculation

4.2 Use of CRESTA-Linear-Solver library

To validate this approach our first step was to implement the conjugate gradient method and generation code for several test sparse matrices. The test cases can be found in the example in the CRESTA SVN repository. The library can be compiled with the GNU, Cray and Intel compilers. Please note, the initial prototype of the library is the basis for further improvement,, testing and linking into real applications. The library needs to be optimized and extended to use it in production mode. As previously mentioned the descriptionof the interface, compilation and runtime parameters of the first public version will be provided in the next deliverable D4.3.2 of work package 4 after the further improvement and validation.

4.3 Further work

The following work is planned to be performed for improving the efficiency and library interface:

- Improve the performance of the synchronization between master and worker threads
- compare the MPI one-sided against MPI_ISend / MPI_IRecv, SHMEM or Coarrays
- Improve the performance of the matrix vector multiplication by use different formats for sub matrices (depending on the number of non zero elements)
- Implement Jacobi (or diagonal) preconditioner
- Implement AMG preconditioner
- Linking of the library into CRESTA applications
- Extension of the library to dynamically adjust the CPU frequency for optimization of power consumption
- Dynamic load balancing through exchange of the sub matrices between the processes

5 Conclusion

The CEL library is in the initial state. The numerical algorithms have been implemented based on the promising new HPC approaches. Nerveless further improvements and optimization needs to be done to achieve the best possible performance. After that we will integrate the library in selected CRESTA applications and use it for some important experiments to test what can be achieved by use of new disruptive technologies.

6 References

[1] Stephen P Booth, Dmitry Khabi, Gregor Matura, Christoph Niethammer, Harvey Richardson.D4.1.1 *Overview of major limiting factors of existing algorithms and libraries.* s.l. : CRESTA Consortium Partners, 2012.

[2] Uwe Küster, Stephen P Booth, Stephen Sachs, Dmitry Khabi, Gregor Matura, Mhd. Amer Wafai.D4.2.1 *Prediction Model for identifying limiting Hardware Factors.* s.l. : CRESTA Consortium Partners, 2013.

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

[4] *An overview of the Trilinos project.* Michael A. Heroux, Roscoe A. Barlett, Vicki E. Howle. s.l. : ACM Press, 2005.

[5] J.A. Åström (CSC), Adam Carter (EPCC),Konstantinos Ioakimidis (USTUTT), Rupert W. Nash (UCL), James Hetherington (UCL), Artur Signell (ABO), Jan Westerholm (ABO).*Needs analysis.* s.l. : CRESTA Consortium Partners, 2012.

[6] Uwe Küster, Dmitry Khabi. Power consumption of kernel operations. *Sustained Simulation Performance.* s.l. : Springer, scheduled at the end of 2013.

[7] Manninen, Pekka.*D4.5.3 – Non-Blocking Collectives Runtime Library.* s.l. : CRESTA Consortium Partners, 2013.

[8] José Gracia, Christoph Niethammer, Wahaj Sethi.*D4.5.2 Microbenchmark Suite.* s.l. : CRESTA Consortium Partners, 2012.