D4.3.2 – Community prototype of exascale algorithms and solver (software)

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:	M38
Submission date:	30/11/2014
Project start date:	01/10/2011
Project duration:	38 months
Deliverable lead organization	HLRS
Version:	1.0
Status	Final
Author(s):	Dmitry Khabi(HLRS), Frederic Magoules (ECP/CRSA)
Reviewer(s)	Jens Doleschal, Michael Wagner (TUD)

Dissemination level	
PU	*PU – Public*

Copyright © CRESTA Consortium Partners 2014

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	20/10/2014	First draft for comments	Dmitry Khabi(HLRS)
0.2	25/10/2014	FFT part	Dmitry Khabi(HLRS)
0.3	26/10/2014	Linear-Solver Part	Dmitry Khabi (HLRS)
0.4	30/10/2014	Domain Decomposition Methods	Frederic Magoules (ECP/CRSA)
0.5	03/11/2014	Performance measurements	Dmitry Khabi (HLRS)
0.6	30/10/2014	Domain Decomposition Methods	Frederic Magoules (ECP/CRSA)
0.7	04/11/2014	Outlook	Dmitry Khabi (HLRS)
0.9	25/11/2014	Addressing all issues from the reviews	Dmitry Khabi (HLRS)
1.0	26/11/2014	Final version for submission	Lorna Smith (UEDIN)

Copyright © CRESTA Consortium Partners 2014

Table of Contents

Index of Figures

Copyright © CRESTA Consortium Partners 2014

Copyright © CRESTA Consortium Partners 2014

1 Executive Summary

This deliverable describes a software package that is a prototype for linear solvers and FFTs intended to run on exascale systems. The software is available on a GitHiub server at:

http://gitlab.excess-project.eu/numlibs/

The CEL library is used within the CRESTA project firstly to support co-design applications in the field of numerical libraries and secondly as a framework for the development and evaluation of some of the other new and promising disruptive technologies, which can be used to improve the efficiency of parallel applications.

© CRESTA Consortium Partners 2014

2 Introduction

This software deliverable is called the CRESTA Exascale Library (CEL). We developed a new library (the CEL) to address two important classes of numerical problem: linear solvers and multi-dimensional Fourier transforms.

In the previous CRESTA deliverables 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) different proposals for the solution of these numerical problems were presented and these have been used as input for the implementation of the library.

The CRESTA Exascale FFT library is intended to make it easier to implement scalable and efficient FFT applications for various data distributions on the network topology.

For the developers of parallel applications, the CRESTA Exascale solver library may help in the efficiency solution of large sparse linear systems.

Section 3 provides describes the solver library while Section 4 describes the various components of the FFT library.

We have carried out research work in the field of another kind of preconditioner, namely flexible asynchronous methods. The research results are published in (3) (4) (5). Although the CEL library does not include those algorithms, the increasing importance of these asynchronous methods led to a disruptive technology workshop in the subject (see, for example the Collaboration Workshop - Edinburgh March 2014 (6)). These methods are discussed in section 5.

© CRESTA Consortium Partners 2014

3 CRESTA-Linear-Solver library

The CEL-Linear-Solver focuses on iterative algorithms to solve large sparse linear systems. The library is being developed using a hybrid approach i.e., a combination of MPI and OpenMP. The source code is written primarily in the programming language Fortran 90 (7). The first version of the community prototype library contains various implementations for the sub operations such as halo data exchange and loop kernels. The distributed matrix-vector multiplication is improved due to the overlapping of computation with communication.

The description of the first public version of the CEL Linear solver is provided in this document. The performance measurements are also presented in this document.

3.1 Compiler and MPI requirements

THE CEL code has been tested for compliance with GNU (from version 4.6), Cray (from version 8.2.5) and Intel (from version. 14.0.2) compilers. The MPI library must support non-blocking collective operations, which were introduced in the MPI 3.0 standard (8). We have compiled the library with the following MPI implementations: Cray MPICH 6.2.X, MVAPICH2 2.0 and MPICH 3.1.

3.2 Data types and constants

The CEL library can be compiled to support any combination of widely used data types. Floating-point numbers can be represented in either single or double precision. The indexes can be represented as 32-bit or 64-bit integers.

We recommend the use of double precision floating-point numbers unless the hardware does not provide support for this. The choice between 32-bt of 64-bit integers depends on the size of the computational problem. In the case of 32-bit integer format the value 2'147'483'647 or (2^{31} -1) is the upper limit for the number of diagonal elements in the corresponding matrix. More details on the influence of data types on the solution and performance are available in (1).

Data types are defined in the configuration file *./make_env.in*. This file provides the option to set various compile time options. The global cel constants are listed in the source file *src/fbase/cel_types_module.f90*.

3.3 Distribution of work and barrier types

As mentioned above the library is being developed using a hybrid approach. The applied MPI_THREAD_FUNNELED thread support defines the roles of the threads (see definition in (8)) within MPI runtime environment:

- The master thread is responsible for the communication between processes and allocation of the shared data structures.
- The worker threads perform the computational work.

To achieve the best balance between computation and communication work we recommend the use of one master thread per NUMA node. Only when utilization of the bandwidth is too low is it advisable to use more than one master thread. Additionally, the quality of the domain decomposition is enhanced due to the reduced number of MPI processes and therefore reduced number of sub-domains.

© CRESTA Consortium Partners 2014

The distribution of the data between the master threads doesn't differ from the standard distribution between the MPI processes in a pure MPI application. The difference between an MPI application and a CEL application is that the data owned by the master will be computed by the worker. The distribution of the data between the worker threads should be done with the subroutine *cel_omp_shared_work_distr* defined in the source file*src/fbase/cel_omp_shared_work_module.f90.*

There are three different computation and communication modes:

- **Exclusive communication**– The master threads exchange the data and the worker threads have to wait until the master releases the updated data.
- **Exclusive computation** –The worker threads compute the new results and the master threads must wait until the workers release the results.
- **Overlapping communication with computation** - The master threads exchange the data and the worker threads compute the already released data.

To synchronize these types of operations we use three different types of barriers: MPI_Barrier, omp barrier and cel barrier. The cel barrier defines the start and end of the region, which is parallelized for the worker threads. The cel barrier is defined in the source file *src/fbase/cel_omp_shared_work_module.f90* according to (9). Figure 1 demonstrates the work of the barriers.

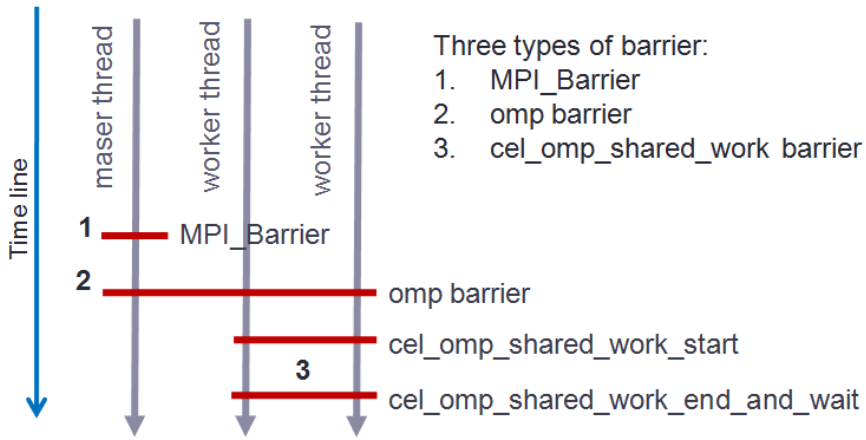

Figure 1 – The three different types of barriers in the cel library: MPI_Barrier, omp barrier and cel barrier.

3.4 Thread affinity

Process and thread placement can be done within the MPI runtime environment. Nevertheless, situations can arise when this is not possible. This can be for a variety of reasons, for example a runtime environment of the system doesn't provide it. In this case, the subroutine for the process and threads placement is included in the library. The *subroutine cel_omp_set_thread_policy* is defined in the source file *src/fbase/cel_omp_module.f90*.

3.5 Numerical algorithms

The first community version contains a set of iterative solvers with a Jacobi preconditioner.

The following numerical methods are implemented in the library:

- Conjugate gradient method (CG, according to the mathematical description in (10)); Defined in the source file *src/fcgalg/cel_cgalg_module.f90*
- Generalized minimal residual method (GMRES, according to the mathematical description in (10));

© CRESTA Consortium Partners 2014

Defined in the source file *src/fcgalg/cel_gmresalg_module.f90*

- Matrix diagonal scaling;
  Defined in the *subroutines cel_sp_mat_distr_scale* and *cel_sp_mat_distr_boundary_diag_scale* in the source file */src/fspmtdistr/cel_sp_mat_distr_format_module.f90*
- Jacobi preconditioner;
  Defined in the *subroutine cel_sp_mat_distr_set_jacobi* in the source file */src/fspmtdistr/cel_sp_mat_distr_format_module.f90*

Currently, more advanced preconditioner are being developed:

- Block Jacoby preconditioner;
- Algebraic multigrid preconditioner, according to the definition in (11);

We have selected the set of algorithms according to the research done in (1). We have also carried out work in the field of another kind of preconditioner, namely flexible asynchronous methods. The research results are published in (3), (4) and (5). The algorithm is also discussed in Section 5. Currently the prototype of CEL library does not include those algorithms because of that complexity.

3.6 Internal sparse matrix format

The overlapping of computation with communication by the matrix-vector multiplication requires some modifications to the data structures for the matrices and vectors. The sparse matrix A is divided into the sub matrices, which are divided into the sub-blocks. The sub matrices are distributed by blocks of contiguous rows. All sub-blocks of the same sub matrix have equal number of rows, which are equal to the length of the local part of the vectors owned by the process and number of rows in the sub matrix. The number of columns is equal to the length of the local part of the vector owned by the corresponding neighbour process. Each of the sub-block matrices can be held in various formats. Document (12) contains more details of the sub-block matrix format. The following formats for the sub-blocks are supported:

- Compressed Row Storage (CRS)
- Modified Jagged Diagonal format (JAD)
- Coordinate format (COO)

The sub-blocks of the matrix are stored in an *allocatable* array of the derived type *cel_sp_mat_type* (defined in the source file *src/fspmt/cel_sp_mat_module.f90)*. The first element of the array is always the diagonal sub-block. The non-diagonal sub-blocks are ordered in accordance with the indexes of the columns.

By default the sub-blocks with the diagonal values are stored in the CRS format. All other sub-blocks are stored in the COO format.

3.7 Halo data exchange

The halo data exchange is one of the limiting factors for matrix-vector multiplication performance. Performance can be enhanced by choosing the most efficient algorithm for the problem and the available hardware. The structure of the matrix, the number of computational nodes, the network topology and the hardware performance influence the choice. The following algorithms are available:

MV_COMM_ISEND is a simple implementation of the halo exchange. In this case the entire part of the local vector is transferred to the neighbours with a set of the asynchronous operation *MPI_Isend*. The disadvantage of this method is that a large part of the transferred data is really not needed for the matrix-vector multiplication.

Defined in the *subroutine cel_sp_mat_distr_vec_isend_start_all* in the source file *src/fspmtdistr/cel_sp_mat_distr_vec_module.f90*

© CRESTA Consortium Partners 2014

MV_COMM_ISEND_IDX is an improved version of the above method. In this case the reduced part of the local vector values are transferred to the neighbours. Nevertheless part of the data transferred is also unnecessary.

> Defined in the *subroutine cel_sp_mat_distr_vec_isend_idx_start_all* in the source file *src/fspmtdistr/cel_sp_mat_distr_vec_module.f90*

MV_COMM_ISEND_COMM_BUFFER transfers only the values that are really needed. The disadvantage of this method is the additional overhead due to indirect addressing and extra data movement between several buffers.

> Defined in the *subroutine cel_sp_mat_distr_vec_isend_comm_buffer_start_all* in the source file *src/fspmtdistr/cel_sp_mat_distr_vec_module.f90*

MV_COMM_IBCAST transfers the entire part of the local vector as in the case of the first method MV_COMM_ISEND. In contrast to other methods, the data will be transferred using non-blocking, with the non-blocking collective operation *MPI_Ibcast* (see more details about that collective operation in (8) and (13)). The halo exchange within small partially disjoint groups may greatly reduce latency. This is illustrated on Figure 2.

> Defined in the *subroutine cel_sp_mat_distr_vec_ibcast_start_all* in the source file *src/fspmtdistr/cel_sp_mat_distr_vec_module.f90*

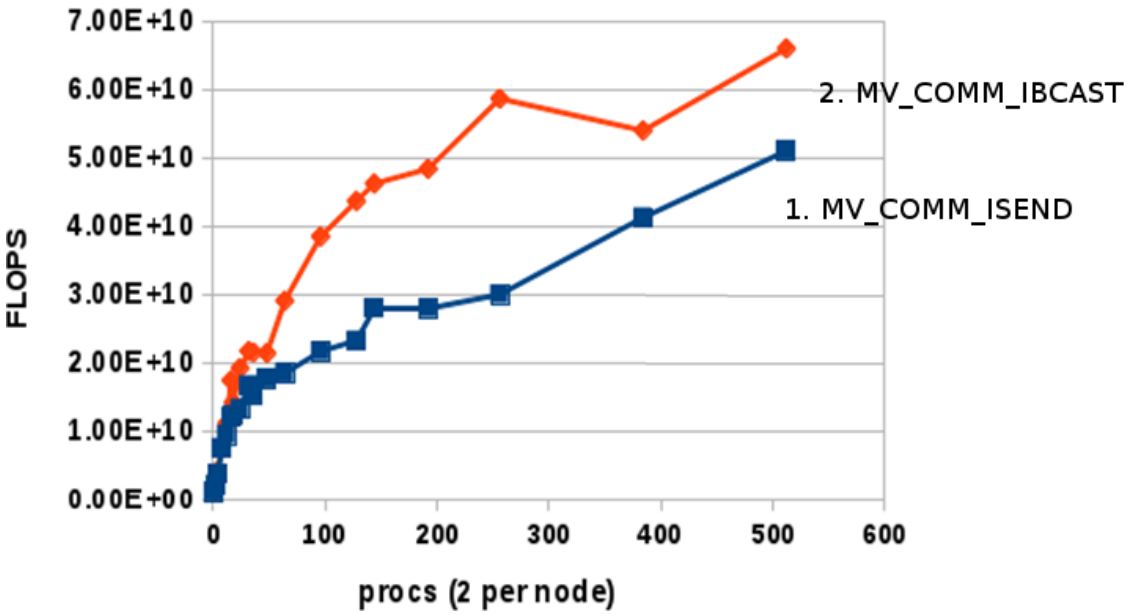

Figure 2 - Total performance of the matrix-vector multiplication depending on the number of MPI processes and the type of the halo data exchange. In the first case the simple method MV_COMM_ISEND was used. In the second case the method MV_COMM_IBCAST for the communication part of the matrix-vector multiplication essentially improved the performance. The measurement was done on the Cray XE6 (Hermit).

Note that in order to use the broadcast operation, a set of the MPI groups must be initialized. The root element of each group is the process that sends the halo data to the neighbours. The remaining part of the group consists of the processes that receive the halo data from the root process. The set of halo data exchange groups is not disjoint since the remainder of the processes obtain the halo data from more than one process as shown on Figure 3.

© CRESTA Consortium Partners 2014

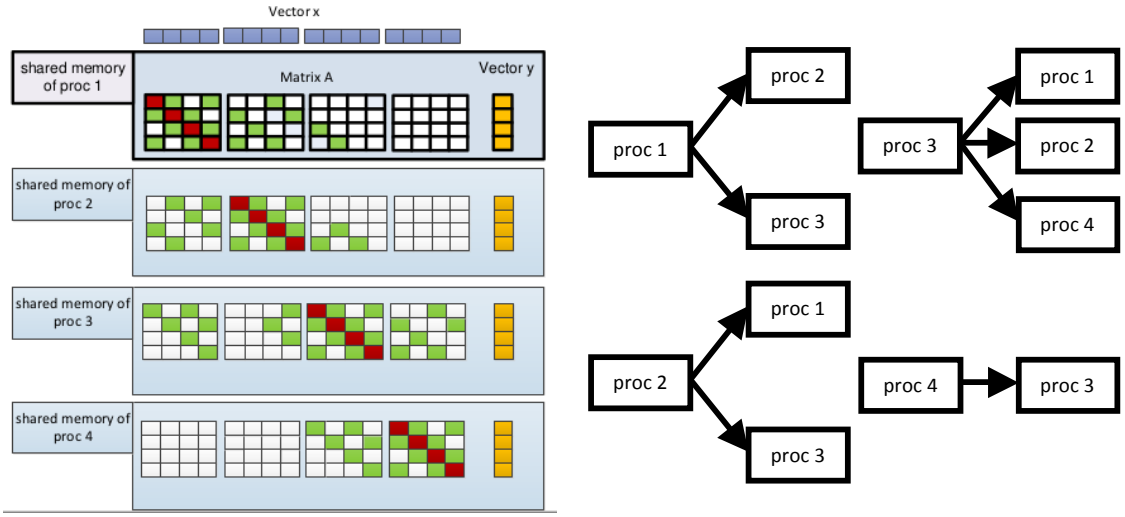

Figure 3 – Left: Distributed sparse matrix in sub-block format. **Right:** The corresponding groups of the halo data exchange with the method MV_COMM_IBCAST based on the MPI non-blocking broadcast operation.

It is advisable to use only scalable MPI functions to create the corresponding communicators (for example, function MPI_Comm_split). Another way is to separate the groups in to several sets with only disjoint groups. The function MPI_COMM_CREATE can be called in parallel for all groups of the set. This algorithm is used in the *subroutine cel_comm_get_communicators* defined in the source file *src/fcomm/cel_comm_module.f90*.

In most test cases the method MV_COMM_ISEND_COMM_BUFFER was the best choice. However, for some special cases the latency reduction may be more efficient than reducing the amount of data that has to be sent and received. This can be true for example, if the sparse matrices have high density and the number of neighbours to communicate is high. Another example is if the bandwidth and the latency of the network are not well balanced (see (12) section 3.4 Network and performance of collective operations for the details).

3.8 Non-blocking and blocking collective operation Allreduce

At compile time the user can also make a choice between blocking and non-blocking version of the collective operation *allreduce*, which is used in the calculation for example to calculate a dot product. The benchmark results on the Cray XE6 and Cray XC30 with cray-mpich2/6.2.1 did not show any significant difference between these two operations. This is shown in the white paper (13). The sections below (3.9, 3.13.1 and 3.13.2) also contain performance measurements of these two collectives operations in the solver.

3.9 Matrix-vector multiplication

The matrix-vector multiplication (*A*x=y*) is one of the most complex and time-consuming operations to solve linear systems. Overlapping computation with communication is one of the simplest techniques to improve its efficiency. Figure 4 shows profile data for the operation. Using 64 cores on 2 computational nodes the halo data exchange was completely overlapped with the local computation. In the case of 256 cores (4 comp. nodes) the execution time is equal to the computation time.

© CRESTA Consortium Partners 2014

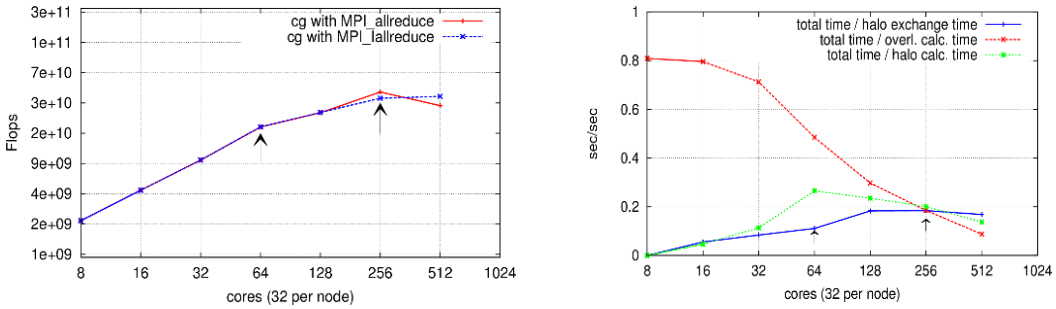

Figure 4– Left: Strong scaling test of the operation matrix-vector multiplication for the Poisson problem on a 3D regular grid of the size 64x64x64 (27 point stencil). The test is performed on the Cray XE6 (Hermit). A maximum four master threads with seven worker threads were placed on each allocated node. **Right**: The normalized execution time of three main sub-operations for the same strong scaling test. The execution time is shown for the halo data exchange (blue curve), overlapping computation (red curve) and computation on the halo data (green). The time was normalized to the total execution time of the matrix-vector multiplication.

The operation is defined in the *subroutine cel_fop_mv_axy_distr* in the source file *src/fop/cel_fop_mv_module.f90*.

3.10 Performance counters

The performance and metadata of various sub-operations will be stored in the cel counters. This allows for performance measurement and for the further integration of auto-tuning into the solvers (changing frequency, activation and deactivation of cores over time). The counter is defined in the source file *src/fbase/cel_perf_module.f90*.

The source file *src/fbasedistr/cel_perf_distr_module.f90* provides methods to gather all counter values from all master processes.

3.11 API interface

There are two interfaces available to link the library. The first interface is for use with a pure MPI program. The schema for this usage is shown in Figure 5.

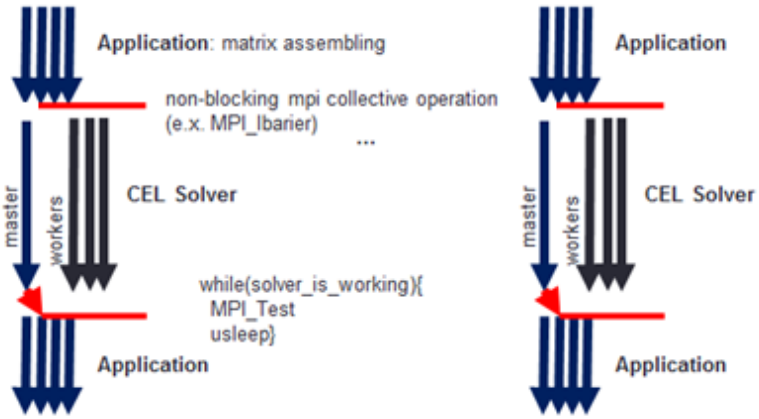

Figure 5 – Linking of the CEL library to a pure MPIMPI application. The N MPI processes on the NUMA node will be reduced to one master process and N-1 worker. Until the solver ends its work the N-1 MPI processes sleep.

The source file *tests/examples/src/cel_solver.f90* contains an example for the usage of that interface.

The source file *tests/examples/src/cel_benchmark.f90* contains the example for the use of the cel library if the application is being developed using the parallel hybrid MPI/OpenMP.

© CRESTA Consortium Partners 2014

3.12 Compilation of the library and examples

After setting all required fields in the configuration file *./make_env.in*, run the make command with the arguments "make clean all tests". This creates a set of libraries in the directory "*./lib"*. The directory *src/include/* contains the Fortran *modfiles*. Two additional examples will be compiled in the directory *./tests/examples/:*

- *cel_solver* – An example of how to use the library interface for a pure MPI application. The matrix for the example must be loaded from the file in the Matrix Market Exchange Format (14). The library includes an example of the matrix in the directory *./tests/matrix/.* The matrix corresponds to a Gyrokinetic code for turbulent fusion plasma of ELMFIRE (15).
- *cel_benchmark* – An example how to use the library interface for a hybrid MPI and OpenMP application. The matrix for the example can be automatically generated or loaded from the file in the Matrix Market Exchange Format.

The arguments for both executables are described in the *README* file.

3.13 Performance measurements of the library

In this section we show performance measurements carried out on two different platforms: a Cray XE6 (Hermit) and a small cluster with dual socket Ivy Bridge E5-2690 v2 processors. A description of the Cray XE6 (Hermit) platform can be found in (16). A description of the Ivy Bridge can be found in (17).

Further performance measurements on Hornet, a new supercomputer Cray XC40 system (18) will be presented after the end of the acceptance phase of the system.

3.13.1 Performance measurements on one node

Figure 6 shows performance results for one node of the Cray XE6 (Hermit). One computation node has two AMD Interlagos processors.

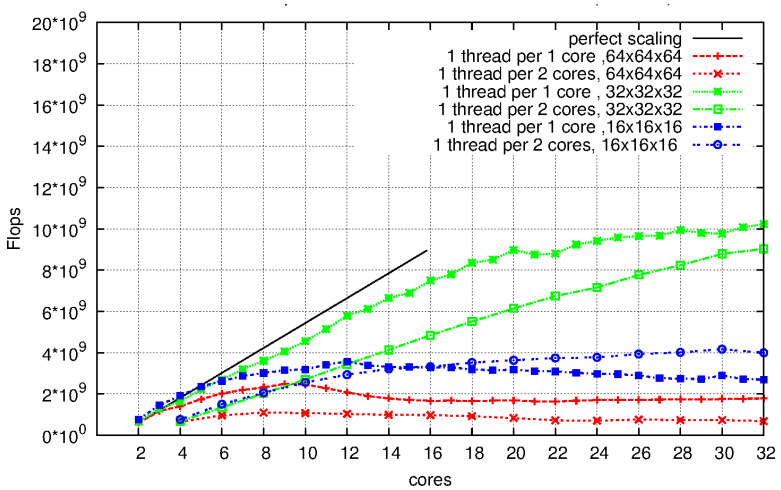

Figure 6 – Performance of the CG for the 3D Poisson problems of sizes 16x16x16x16, 32x32x32 and 64x64x64. The tests were performed on one computation node with two AMD Interlagos processors. Each process has 16 cores. Only one master process was started.

Figure 7 shows the performance of the CG solver on the computation node with two Ivy Bridge processors.

© CRESTA Consortium Partners 2014

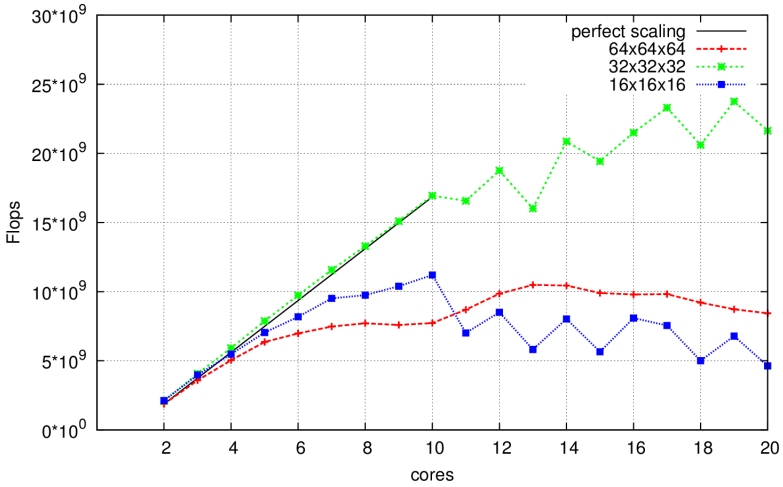

Figure 7 - Performance of the CG for the 3D Poisson problems of sizes 16x16x16x16, 32x32x32 and 64x64x64. The tests were performed on one computational node with two Ivy Bridge E5-2690 v2 processors. Each process has 10 cores. Only one master process was started.

3.13.2 Weak scaling on the Cray XE6 (Hermit)

Weak scaling tests were carried out on the Cray XE6 (Hermit).

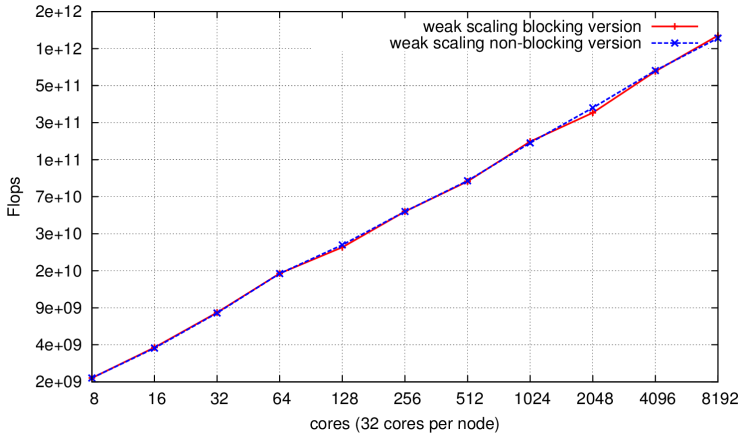

Figure 8 Weak scaling test of the CG with Jacobi preconditioner for the Poisson problem on a 3D regular grid (27 point stencil). The first run was carried out with one master process and seven worker threads. The matrix had 64x64x64 rows.

© CRESTA Consortium Partners 2014

4 CRESTA-FFT library

We developed a generic library (reshape) to support changes in the data decomposition that can be used to quickly optimize the FFT strategy with respect to the hardware.

Our approach emphasizes the communication component of the FFT problem. This kind of communication seems to be one of the scaling issues for exascale computing.

In general any data decomposition can be represented at runtime 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 dataset 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 drawback is that these general decompositions are a bit 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 data types corresponding to the necessary messages). Any additional overhead only takes place in the initial planning stage and should have a small 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 data types to support parallel IO to the different decompositions.

A detailed overview of CRESTA-FFT library is given in the CRESTA deliverable D4.3.1 "*Initial prototype of exascale algorithms and solvers for project internal validation*" (12) so that will not be fully repeated here. The source code includes examples of how to use the library.

© CRESTA Consortium Partners 2014

5 CRESTA domain decomposition methods

The traditional scheme for parallel iterative algorithms like the CRESTA-Linear-Solver library is synchronous iterations. This describes a method where a new iteration can only start when all data of the previous iteration has been received. These parallel synchronous iterations use the same mathematical model and have the same convergence behaviour as their sequential counterparts. They have been widely studied and are often simply called parallel iterative algorithms, synchronous being omitted. They are very efficient when used with well-balanced workloads and short communication times. However, these conditions are hard and expensive to achieve especially as the number of cores used increases.

Another kind of iterative algorithm can help to solve these scalability problems. Called first chaotic relaxations, it is now usually designed by asynchronous iterations. There are three commonly used schemes for asynchronous iterations. The first one is totally asynchronous iterations. It sets very few conditions on the iterations and communications except that they must never stop indefinitely. The second scheme is partially asynchronous iterations. It is based on the assumption that the communication time is bounded and that each process does at least one iteration every given period. The third scheme is flexible asynchronous iterations. Flexible communication means that the data is sent as soon as possible, but due to the strength of asynchronous iteration, it is more general.

Since iterative algorithms are often slow to converge, more advanced algorithms like domain decomposition methods must be considered. The extension of such algorithms to totally asynchronous iterations leads from a mathematical point of view to several convergence problems and from a computer science point of view to define new parallel paradigms. The first results of the extension to totally asynchronous iterations of the optimized Schwarz algorithm show that this algorithm is extremely robust compared to classical iterative algorithm, and very promising for Exascale computing. But the optimized coefficients must be derived upon the mathematical equation of the physical problem considered.

In order to implement an asynchronous iterative algorithm, an additional layer between the MPI library and the code must be developed. This layer will in particular allow in the definition of asynchronous communications between the processors, with a continuous request, and to define the best way to evaluate the residual to determine the convergence of the algorithm. Once this layer is defined, the code itself must be slightly modified as well. This layer-code modification must be performed manually for each algorithm considered.

© CRESTA Consortium Partners 2014

6 Outlook

The CEL library is publicly available. The library will be further developed, for example in the project “EXCESS” (19) in order to improve the efficiency of the library. This will be achieved by including more a advanced preconditioner for the scalability, matrix vector blocking techniques for better loop vectorization and dynamic changes to the number of the threads depending on the load balance and type of computational kernels. Frequency scaling is also a promising disruptive technology as shown in early CRESTA work (see (20) for more details). The *cel_omp_shared_work_module* will be used to manage the number of computing threads and their frequency depending on the metadata, which is collected in the cel counters.

© CRESTA Consortium Partners 2014

7 Bibliography

1. **Stephen P Booth, Dmitry Khabi, Gregor Matura, Christoph Niethammer, Harvey Richardson***Overview of major limiting factors of existing algorithms and libraries*CRESTA Consortium Partners2012

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

3. Asynchronous Schwarz methods for peta and exascale computing.*In B.H.V. Topping and P. Ivanyi, editors, Developments in Parallel, Distributed, Grid and Cloud Computing for Engineering, chapter 10, pages 229-248.*Stirlingshire, UKSaxe-Coburg Publications2013

4. Asynchronous optimized Schwarz methods.*In B.H.V. Topping, editor, Computational Methods for Engineering Science, chapter 17, pages 425-444*Stirlingshire, UKSaxe-Coburg Publications2012

5. **C. Venet and F. Magoules**Asynchronous substructuring methods.*In Substructuring Techniques and Domain Decomposition Methods, chapter 4, pages 71-104*Stirlingshire, UKSaxe-Coburg Publications2010

6. Collaboration Workshop - Edinburgh March 2014CRESTA Consortium Partners 2011http://www.cresta-project.eu/news-events/collaborative-workshop.html

7. Fortran Standards Documents: Fortran 90. [Online] https://gcc.gnu.org/wiki/GFortranStandards.

8. *MPI: A Message-Passing Interface Standard Version 3.0 Chapter author for Collective Communication, Process Topologies, and One Sided.* s.l. : Message Passing Interface Forum. , Sep. 2012.

9. Locks and Barriershttp://www.it.uu.se/edu/course/homepage/os2/st09/handout-04.pdf

10. *Numerik linearer Gleichungssysteme*Friedr. Vieweg & Sohn Verlag2008ISBN 978-3-8348-0431-0

11. *Algebraic Multigrid (AMG): An Introduction with Applications*

12. *D4.3.1 – Initial prototype of exascale algorithms and solvers for project internal validation*CRESTA Consortium Partners 20112013

13. *White Paper Benchmarking MPI Collectives at SC14.* **Christoph Niethamme, Pekka Manninen, Rupert Nash, Dmitry Khabi, Jose Gracia.** s.l. : CRESTA Consortium Partners, 2014.

14. Matrix Market Mathematical and Computational Sciences Division of the Information Technology Laboratory of the National Institute of Standards and Technology.http://math.nist.gov/MatrixMarket/formats.html

15. *Full-f gyrokinetic method for particle simulation of tokamak transport* **J.A. Heikkinen, S.J. Janhunen, T.P. Kiviniemi and F. Ogando**Journal Comput. Phys.227 (2008) 5582-5609.

16. Cray XE6 (HERMIT)https://wickie.hlrs.de/platforms/index.php/Cray_XE6

17. Intel® Xeon® Processor E5-2690 v2 (25M Cache, 3.00 GHz)http://ark.intel.com/products/75279/Intel-Xeon-Processor-E5-2690-v2-25M-Cache-3_00-GHz

18. CRAY XC40 (HORNET)HLRShttp://www.hlrs.de/systems/platforms/cray-xc40-hornet/

19. Execution Models for Energy-Efficient Computing Systems - EXCESSEXCESS Partnershttp://excess-project.eu/

© CRESTA Consortium Partners 2014

20. **Alistair Hart, Michele Wieland, Dmitry Khabi, Jens Doleschal***D2.6.3 – Power measurement across algorithms*CRESTA Consortium Partners 20112014

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

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

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

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

25. **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.

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

27. EXASOLVERS - Extreme scale solvers for coupled problemsDFGhttp://www.sppexa.de/general-information/projects.html#EXASOLVERS

28. *D4.5.3 – Non-Blocking Collectives Runtime Library*CRESTA Consortium Partners 20112013

© CRESTA Consortium Partners 2014