E-Infrastructures

H2020-EINFRA-2015-1

EINFRA-5-2015: Centres of Excellence for computing applications

EoCoE

Energy oriented Center of Excellence for computing applications

Grant Agreement Number: EINFRA-676629

D6.14 M24

D 6.14 Report on knowledge transfer for co-design of exascale architectures

Project and Deliverable Information Sheet

EoCoE	Project Ref:	EINFRA-676629
	Project Title:	Energy oriented Centre of Excellence
	Project Web Site:	http://www.eocoe.eu
	Deliverable ID:	D6.14 M24
	Lead Beneficiary:	CEA
	Contact:	Edouard Audit
	Contact's e-mail:	edouard.audit@cea.fr
	Deliverable Nature:	Report
	Dissemination Level:	CO*
	Contractual Date of Delivery:	M24 30/09/2017
	Actual Date of Delivery:	M32 30/04/2018
	EC Project Officer:	Carlos Morais-Pires

* - The dissemination level is indicated as follows: PU – Public, CO – Confidential, only for members of the consortium (including the Commission Services) CL – Classified, as referred to in Commission Decision 2991/844/EC.

Document Control Sheet

Document	Title :	D 6.14 Report on knowledge transfer for co-design of exascale architectures
	ID :	D6.14 M24
	Available at:	http://www.eocoe.eu
	Software tool:	Microsoft Word
Authorship	Written by:	Paul GIBBON (JUELICH)
	Contributors:	Dirk PLEITER, Wendy SHARPLES, Matthieu HAEFELE, Pasqua D'AMBRA and Salvatore FILIPPONE
	Reviewed by:	George KIRKOS, Edouard AUDIT, Nathalie GIRARD, PEC members

Contents

1. Introduction 3

2. Evaluation of relevant technology and market trends 4

2.1. Processor market 4

2.2. Compute accelerators 5

2.3. Memory and storage technologies 6

2.4. Analysis of impact on scientific computing 7

3. Identification of exascale/co-design candidates 7

4. Development, testing and release of Mini Apps 8

4.1. ParFlow 8

4.2. Metalwalls on FPGA 10

4.3. GPU plugins for PSBLAS and MLD2P4 12

5. Developing generalized performance characterization metrics 14

5.1. ParFlow follow-up POP analysis - case study 14

6. Compiling and establishing suitable definition of requirements with reference to hardware architecture and programming models 16

7. Development of performance models 17

8. Evaluating performance and designing balance for different future architectures 18

1. Introduction

Although originally intended as part of the WP6 activities, much of this work has naturally arisen out of activity performed in WP1. Accordingly, the present report is structured as follows: first in section 2, we present a general survey of technology trends in the HPC landscape. This is followed by a short summary of the work already reported on identifying exascale candidate applications out of the many codes examined at the EoCoE-POP workshops and during follow-up activities. Specific examples of work with mini-apps on future hardware technologies is documented in section 4, and followed by descriptions of the metric definitions and contributions to hardware requirements for the SRA-3 exercise in 2017. Finally, we conclude with a list of EoCoE contributions to contemporary FET-HPC projects.

2. Evaluation of relevant technology and market trends

2.1. Processor market

During the last decade the HPC market started to be dominated by a single processor technology, namely Intel Xeon. The number of systems listed in the Top500 list[1] using different Intel Xeon processors increased from 371 in June 2012 to 457 in November 2017, i.e. more than 90% of the listed systems used this technology. However, new processor technologies are emerging, which could challenge this situation.

Two large-scale systems based on IBM POWER9 processors are currently being built-up in the US at Oak Ridge National Lab (ORNL) and Lawrence Livermoore National Lab (LLNL). This processor technology attracted new interest due to its enhanced capability of integration with GPUs thanks to the NVLink technology. AMD has introduced a new generation of processors under the name EPYC. They differ from the newest generation of Intel Xeon server processors (Skylake) by providing a larger number of cores (up to 32 instead of 28[2]), less wide SIMD units (256 bit instead of 512 bit) and a larger number of DDR channels (8 instead of 6). This results in a smaller ratio between raw throughput of floating-point operations versus raw memory bandwidth. Cavium is introducing an ARMv8 based processor called ThunderX2 with similar characteristics.[3] The theoretical memory bandwidth of the ThunderX2 processors is due to the 8 DDR channels similar as for the AMD EPYC processor.[4] Due to the only 128 bit wide SIMD units the throughput of floating-point operations is much smaller. However, as more and more applications start to become memory bandwidth limited on modern processor architectures, initial performance figures for this processor architecture indicates it being competitive with Intel's current Xeon processors. In future the situation will change, once the new Scalable Vector Extension (SVE)[5] ISA, which was announced by ARM in 2016, is implemented by processor manufacturers. SVE would allow for vectors with a length of up to

1 https://www.top500.org/statistics/list/

2 https://ark.intel.com/products/codename/37572/Skylake

3 https://www.cavium.com/product-thunderx2-arm-processors.html

4 https://www.amd.com/en/products/epyc-7000-series

5 https://developer.arm.com/products/software-development-tools/hpc/sve

2048 bits, although currently only implementations with 512 bit have been announced.[6]The Japanese post-K supercomputer will be based on the ARM SVE technology.

In 2017 Intel announced a discontinuation of its Xeon Phi product line. These processors had been optimised for very large count (up to 72 cores in the latest generation) of simple cores, which feature a high throughput of floating-point operations due to two 512 bit wide SIMD units. This step indicates challenges of processor manufacturers to bring processors to market providing a raw throughput of floating-point operations in the beyond 2 TFlop/s per processor range.

There are, however, exceptions as demonstrated for the system, which is listed as fastest system since the June 2016 Top500 list. The Chinese Sunway TaihuLight system is based on a custom chip with 256 compute and 4 control (master) cores. In combination with a 256 bit wide vector unit per core, it reaches almost 3 TFlop/s at a core clock frequency of only 1.45 GHz.

2.2. Compute accelerators

This development of processor technologies is expected to increase the interest in architectures comprising accelerators for floating-point computations. The currently most widely used solutions are graphics processors (GPUs). The most recent generation of NVIDIA GPUs, which is called Volta, can deliver up to 7.5 TFlop/s.[7] The performance of the aforementioned systems Summit[8] and Sierra[9] at ORNL and LLNL depend significantly on the performance of these accelerators.

Changes in the market of semiconductor manufacturers make it easier for non-established hardware producers to realised devices based on advanced CMOS technologies. It is expected that this will lead to more domain-specific hardware, which will typically be accelerators. A well-known example for this development is Google's Tensor Processing Unit (TPU),[10] which is used for accelerating the inference phase of Deep Learning applications.

Another technology that is being explored for use as accelerator of scientific computing applications are FPGAs. This technology is of interest as it may allow for a significant improvement of energy efficiency and could be a path towards hardware specialisation to realise performance improvements. In the context of a Pre-Commercial Procurement (PCP), which was executed within the PRACE-3IP project[11], the British SME Maxeler could demonstrate that their technology start to make porting of complex scientific applications to

---

6 Fujitsu announced to develop a processor with 512 bit wide SVE units for the Japanese Post-K computer (http://www.fujitsu.com/global/Images/armv8-a-scalable-vector-extension-for-post-k.pdf).

7 This is the performance with boost clock enabled. NVIDIA has not yet disclosed the performance at default clock. For more information see: http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

8 https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/

9 https://computation.llnl.gov/computers/sierra

10 Norman P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," ISCA 2017 conference, 2017 (doi:10.1145/3079856.3080246).

11 See for instance: http://eafip.eu/wp-content/uploads/2016/11/2_P.Segers.pdf

FPGAs feasible.[12] However, the porting efforts are still considerable and the precision, at which the floating-point operations are being performed, might need to be changed. Therefore, more efforts are needed to demonstrate the usability and benefits of this technology.

2.3. Memory and storage technologies

Memory technologies is another area, which is expected to have significant impact on future HPC architectures. Most HPC systems as of today provide at the node level a single tier of volatile memory based on DDR SDRAM. This technology continues to improve in terms of capacity and, at a much slower path, in terms of bandwidth. Given the significant growth in compute performance, compute node architectures have often become unbalanced in terms of compute versus memory performance. Furthermore, the lacking drop in price per capacity made it difficult to afford building systems with a larger overall memory capacity. In June 2012 the Japanese K Computer with a memory capacity of 1.34 PiByte was first listed in the Top500 list. The aforementioned Sunway TaihuLight system provides a memory capacity of only 1.25 PiByte despite an 11-fold increase of raw throughput of floating-point operations.

As a consequence of this development it is to be expected that in future different memory technologies will be integrated in HPC architectures. The resulting multiple memory tiers can be optimised for capacity or performance. The future Summit system at ORNL will be based on compute nodes with three different types of memory technologies: high-bandwidth memory integrated in NVIDIA Volta GPUs, DDR4 memory as well as non-volatile memory devices attached to IBM POWER9 processors. The target memory footprint is more than 10 PiByte, with most of the bandwidth and capacity being provided by the GPU memory and the the non-volatile memory, respectively. The increased deepness of the memory hierarchy combined with the necessity of explicit data transfer between non-volatile memory and the SDRAM/GPU memory remains a challenge for their efficient usage in the context of scientific applications development.

Storage architectures face a similar challenge as discussed above for memory technologies. Also for typical storage technologies like hard-drives, capacity grows faster than performance. Therefore, also here the introduction of multi-tier architectures is taking place. The realisation of so-called burst buffers[13] is a first step in this direction.

I/O performance of scientific applications is, however, not only limited by the bandwidth to the I/O subsystem. Also the performance of metadata operations can cause a bottleneck. This limitation is mainly caused due to applications mandating a POSIX-compliant interface to storage. New objects storage technologies like Mero[14] allow to overcome this problem. The benefits of adopting I/O interfaces that do not rely on POSIX and the willingness of the scientific community to endorse these, still remain to be established and explored.

---

12 In the context of the EoCoE there are ongoing efforts to port the application Metalwalls to the pilot system deployed by Maxeler at JUELICH.

13 Ning Liu et al., “On the Role of Burst Buffers in Leadership-Class Storage Systems,” MSST 2012, 2012 (doi:10.1109/MSST.2012.6232369).

14 Nikita Danilov et al., “Mero: Co-Designing an Object Store for Extreme Scale,” PDSW-DISCS 2016 (http://www.pdsw.org/pdsw-discs16/wips/danilov-wip-pdsw-discs16.pdf).

2.4. Analysis of impact on scientific computing

The envisaged changes in key technologies for today's and future HPC systems will potentially have significant impact on the developers of scientific computing applications. Exploiting these technologies efficiently will require significant development efforts. Collaboration of FETHPC projects as listed in section "Relation to ongoing FETHPC projects" can help to address these challenges.
In future, a broader variety of processor architectures based on different Instruction Set Architectures (ISA) might be used. The parallelism within these processors will increase, but details in terms of number of cores, threads per core or vector length will differ. The work of projects like MontBlanc-3 or MB2020 addressing currently available or future ARM-based processor architectures are of interest in this context.

A significant increase in compute performance is likely requiring the use of accelerators resulting in heterogeneous compute node architectures. Also due to the expected use of different types of memory technologies, heterogeneity at node and system level is expected to increase. Results of projects as ALLLScale, ANTAREX or INTERTWInE can help to hide some of the complexity and facilitate the development of performance portable code.

New non-volatile memory technologies also allow to improve performance of storage subsystems at the price of these becoming more complex due to multi-tiering. Projects as NextGENIO and SAGE try to facilitate both, the integration of these non-volatile memory technologies as well as making the resulting architectures usable. How EoCoE applications can exploit the results of these projects needs still to be explored, as these only start to become available.

3. Identification of exascale/co-design candidates

Although a number of highly scalable codes were known to the EoCoE WP1 team prior to the start of the project, a truly quantitative assessment of readiness was only made possible through the joint EoCoE-POP benchmarking workshops (3 so far), which clearly marked out a set of codes falling into this category. Figure 1displays a selection of applications which were examined at at least one of the workshops and where follow-up work led to significant improvements in performance. These include ALYA, PARFLOW, Gysela and the ensemble framework ESIAS, which also formed the basis of the proposal for the successor project EoCoE-II. During the process of building the latter project, it was found that this subset of applications naturally generated a series of technical challenges which have to be resolved in order to push them to exascale performance levels. This will also involve stronger engagement with future European hardware initiatives within ETP4HPC and EuroHPC (see section 8.)

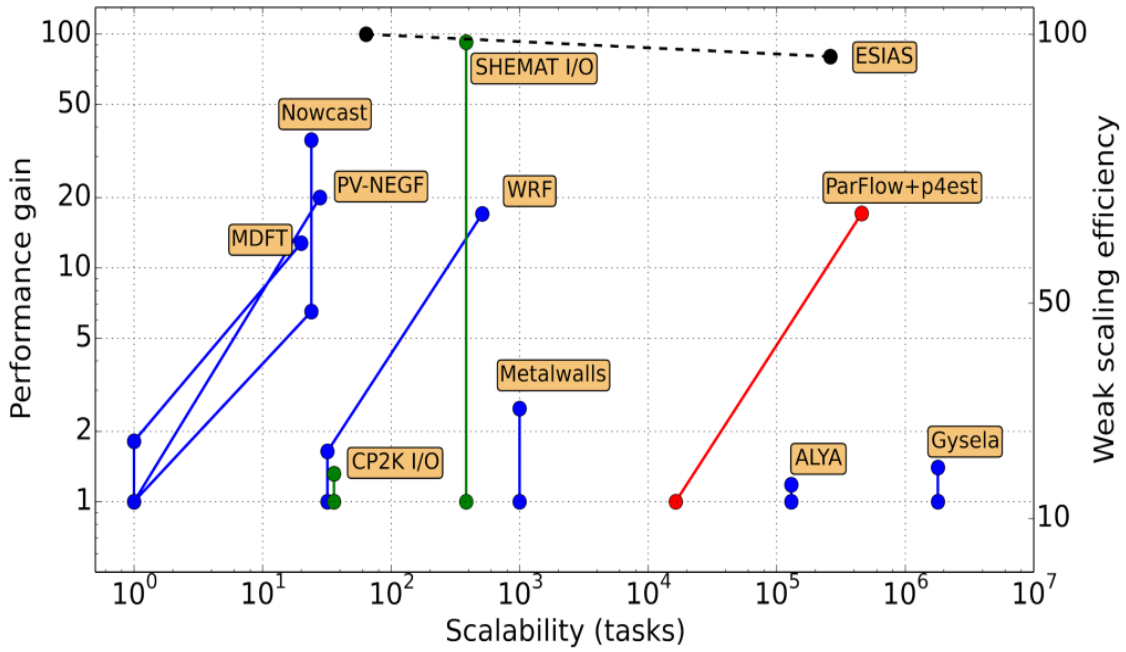


Figure 1. EoCoE application improvements since the first benchmarking workshop

4. Development, testing and release of Mini Apps

4.1. ParFlow

The ParFlow application was analysed in close collaboration with scientists and technicians from JUELICH and PSNC and the performance of ParFlow + PETSC benchmark was evaluated. A new mini-app was created and prepared in order to test PETSc for several different configurations and problem sizes. The mini-app is based on a Python script taking advantage of pets4cpy module (built as a python's wrapper for the PETSc library). PETSc itself permits a choice of matrix-solver method (e.g. mpiaij, mpiaijcusparse), where it can be explicitly decided whether data will be placed on CPU (mpiaij) or GPU (mpiaijcusparse). The script can be summarized as follows:

```
A = PETSc.Mat().create(comm=comm)
A.setType(PETSc.Mat.Type.MPIAIJCUSPARSE)
```

With respect to I vector it can be done with:

```
b.setType(PETSc.Vec.Type.MPICUSP)
```

Finally, A and b can be loaded and the problem solved.

```
A.load(viewer)
b.load(viewer)
```

Via the communicator a choice can be made whether all operations are done in serial or parallel way. For this example, mostly due to size of input data the MPI communicator was chosen

```
comm = MPI.COMM_WORLD
```

The mini-app can be used in both read- and write-mode, where the latter lets the user generate matrices. The matrices used to solve the equation Ax = b were generated based on random permeability matrix and an input pressure matrix with a source and a sink. The permeability values are based on a log normal distribution where the standard deviation is set from 1 to 3; 1 meaning the least amount of heterogeneity and 3 being the most. The permeability matrix is then tridiagonalized to reduce the number of arithmetic operations to become the A matrix used in Ax = b solve. The read-mode of the mini-app is purely dedicated to solve the problem based on previously generated input data.

For test-purposes, initial tests were carried out on an Eagle cluster for the problem size of 200^3 and 224^3, yielding matrix sizes of 8000000^2 and 11239424^2 respectively. Of course corresponding sizes of the sparse matrix are much smaller. Finally, mostly because of future usage the application in the scope of energy consumption measurements using the Score-P analysis tool and some problems with python integration, it was decided to take a pure C-version for solving purposes and use the Python script solely to generate the data.

Following the above procedure, a set of tests were executed to select the best promising method for solving linear system of equations in the field of hydrology. This involved several iterative solvers and their setups as well as several hardware architectures available on JSC and PSNC:

Partition architecture type	Vendor/chip
Cluster	Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
Booster	Intel(R) Xeon Phi(TM) CPU 7250F @ 1.40GHz
GPU	Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz + 4 x NVidia Tesla K80
ARM	ARM A72 prototype @ 2.5GHz - tests are not completed yet

Additionally, test scenarios included both strong and weak scalability runs for selected methods available from Krylov space methods. These methods are considered as one of the ten most important classes of numerical method especially in large sparse linear systems of equations or large sparse matrix eigenvalue problems. The general conclusions from all performed tests are:

1. Stronger heterogeneity (num = 3) of permeability matrix improves strong scaling
2. The problem size of 160^3 and 80^3 for GPUs is too small. In best setups, it scales well up to 16 and 8 tasks; most probably dimensions of permeability matrix should be increased
3. For the scalable parts, then for GPUs as well as CPUs:
    a. cg is always better than bcgs
    b. for cg method, mg-na is worse than bjacobi with ilu,gmres,Jacobi
    c. the three best algorithms appear to be: cg-bjacobi-{ilu,gmres,jacobi}
4. CPU strong scaling for the "160"-grid goes well up to 64 tasks;

The graphs below in Figure 2 below present the performance and scalability of various methods of the problem solving on different platforms: a) and b) GPU c) Intel Xeon d) Intel PHI e) ARM-72 f) comparison of all platforms

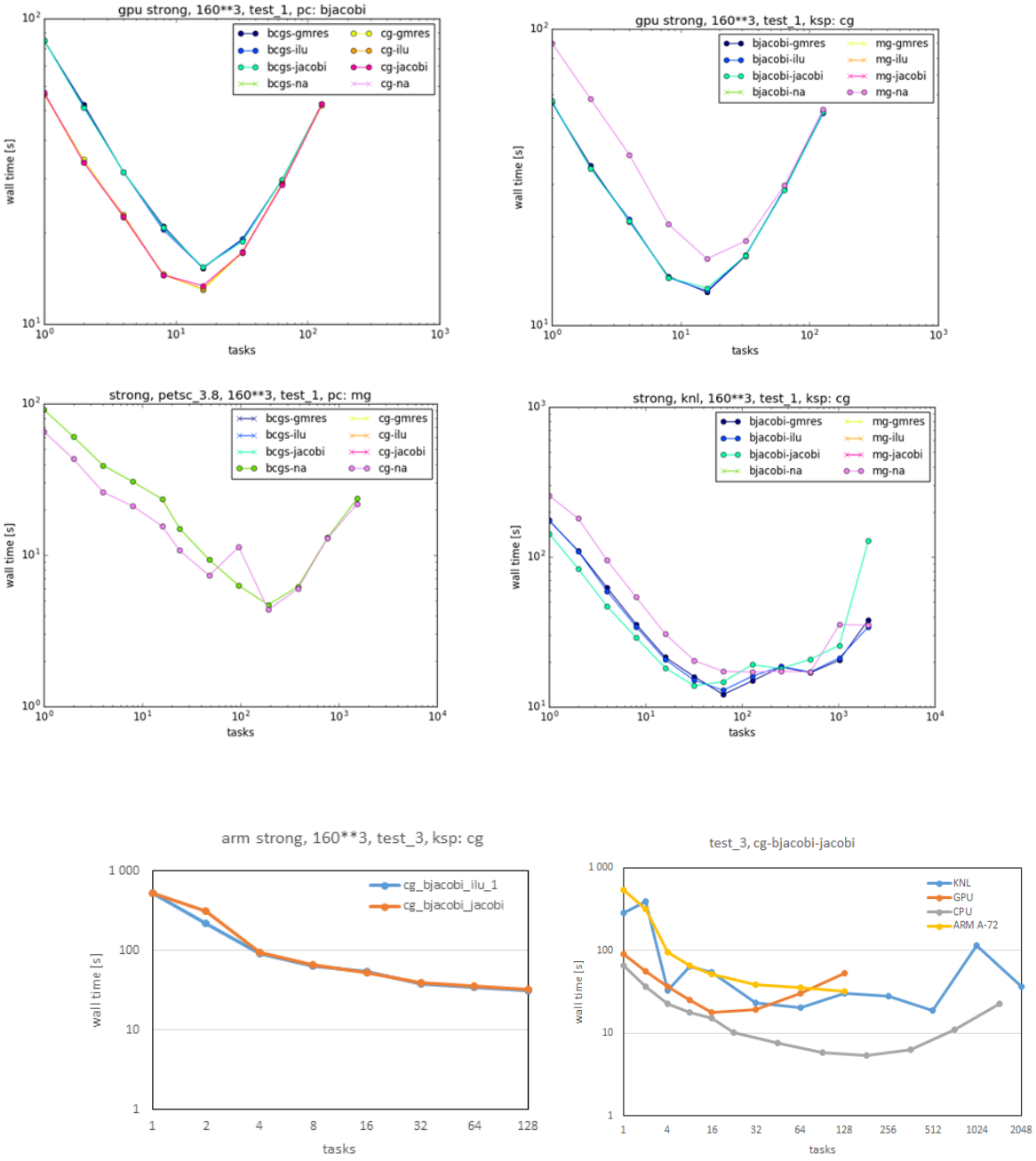


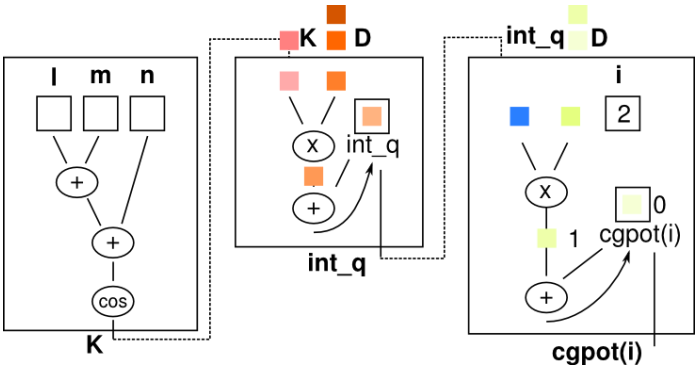
Figure 2. Tests with ParFlow mini-app

4.2. Metalwalls on FPGA

Some efforts have been made to port Metalwalls, or at least a part of it, on FPGA, more specifically on the Data Flow Engine (DFE) technology developed by Maxeler. Matthieu Haefele was invited by Maxeler to spend two weeks in their office in London to get started on this new technology.

The development of a DFE implementation requires a change of paradigm in code design compared to CPU or even GPU programming. Indeed, on a CPU or a GPU, data movements are triggered by the flow of instructions of the application and are then handled by the

hardware. The intrinsic data stream character of a FPGA leaves to the developer the care of moving the data explicitly. Multiply, add or more complex operations like cosine or other complex functions are then put on the different data paths in order to implement the desired algorithm. Figure 3 shows the DFE implementation of kernel and the original fortran code. Three DFE kernels are necessary to increase computations concurrency due to reduction operations. Lines inside and between kernels are data paths inside the FPGA chip on which the different operations are put. The colored squares represent data flowing inside the chip along these paths. The array D is streamed in the chip from memory by two different kernels and the cgpot array, the result of the algorithm, is streamed out back into memory.




```
cgpot = 0.0
do l = 0, nb_mode_x
   do m = -nb_mode_y, nb_mode_y
      do n = -nb_mode_z, nb_mode_z
         int_q = 0.0
         K = cos(l+m+n)
         do i = 1, Nmelt + Nwalls
            int_q += K*D(i)
         end do
         do i = 1, Nwalls
            cgpot(i) += int_q*D(i)
         end do
      end do
   end do
end do
```

Figure 3. DFE implementation of the kernel presented on the right

Knowing the amount of data that should go through each DFE kernel, the number of required clock ticks to perform the full kernel execution is known. As a consequence, the performance of the implementation can be predicted with a very good confidence (5-10%) with a simple spreadsheet. So the design phase of the DFE implementation is strongly driven by this spreadsheet analysis. Once the design finished, the implementation can start using MaxJ, the embedded DSL based on java developed by Maxeler. It allows to describe how data flows within each kernel and how kernels are connected together. An eclipse plug-in speeds up the development and provides a comfortable unit testing environment for each kernel.

As a compilation that would allow to run the application on the real hardware takes between 24 and 48 hours, all executions needed to develop the algorithm are performed within a FPGA simulator/emulator in order to check the algorithm correctness. A basic emulator is used to run the unit tests but cannot execute algorithms that require more than one kernel. The full emulator has then to be used in this case with the offload mechanism. Small fortran proxy applications have been developed to read input files, initialise data structures and call the right routine in order to offload the desired workload onto the FPGA. The emulator is then triggered, it executes the kernels and sends back the result to the fortran application. The full algorithm correctness can be then tested this way provided the test case is small enough to fit and to be run in a reasonable time within the emulator.

Porting Metalwalls to FPGA consists then in implementing DFE kernels to run the most computing intensive part of the code on the device. In term of code base, this represents 30%

of the 20k lines of the full code. A subset of these 30% is currently targeted for this activity, namely the conjugate gradient part. If this part can be successfully and efficiently ported to FPGA, there is a good chance that the full 30% can be ported and ensure maximum performance as the communication between the CPU and the FPGA will be minimum. We have currently successfully set up a design that tells us that a single FPGA should be faster than a bi-socket Haswell node by a factor 4-5. The corresponding kernels have been implemented, tested in the emulator and they give the same result as the original application on a very small test case. We are at the point where a run on the real hardware is mandatory. This requires a compilation and then the support from Maxeler to achieve this as it is a complex task and is completely new for our team. The natural computing system to perform the runs is the PRACE PCP machine based on Maxeler technologies hosted at JSC. Accounts on the machine have been created for project members and we are eager to measure run time and energy requirement.

4.3. GPU plugins for PSBLAS and MLD2P4

Some efforts of University of Rome Tor-Vergata and CNR are devoted to extend the current version of PSBLAS and MLD2P4 with GPU plugins, in order to efficiently run the available solvers and preconditioners on current GPGPU accelerators. The main activities have been devoted to improve an existing GPU plugin for efficient sparse matrix-vectors computations on NVIDIA GPUs. In particular, close attention was devoted to the creation of a convenient interface that plugs some inner kernels, such as sparse matrix-vector multiplication specifically tuned for GPUs, into the PSBLAS framework by employing some modern software engineering techniques, e.g., design patterns. The most important issue was the development of run-time support for different storage schemes for sparse matrix: it is well known that data storage formats are essential for efficiency of sparse matrix computations, due to their non regular data access pattern. Different processors are best exploited by different formats, for example a version of the Ellpack standard format which stores nonzero entries of a sparse matrix into a regular array structure where each column stores the nonzeros of each matrix row, is a good candidate for implementing sparse-matrix operations on GPUs, while the standard CSR data storage scheme appears inefficient. On the other hand, efficient usage of heterogeneous architectures could require to change data storage formats at compile time and run time in response to machine changes and usage requirements and, furthermore, user's interface of kernels should be (almost) independent of the target machine.

To give efficient answers to these issues, the group of the University of Rome “Tor-Vergata” employed some standard design patterns in Fortran 2003 to develop an efficient plugin for PSBLAS to do main sparse matrix computations, including all the main kernels of a Krylov solver, on GPUs. The above techniques have been also used to design an interface between PSBLAS and the CuSparse library, the Nvidia Cuda library which provides a collection of basic linear algebra subroutines used for sparse matrices. Further activities are carried on the package MLD2P4, which implements a set of parallel preconditioners based on Algebraic MultiGrid methods (AMGs) on the top of PSBLAS. In detail, some internal changes have been applied to MLD2P4 to guarantee optimal use of the GPU plugin available from PSBLAS and a plugin for computing different versions of sparse approximate inverses well suited as smoother and solver on GPUs is currently under testing on the EoCoE applications. AMGs derive their efficiency by a recursive application of a two-grid process which consists of some

smoother iterations and a coarse-grid correction. Computational kernels of AMGs includes sparse matrix/vector multiplications and vector updates for moving among the grids, whose efficient parallel implementation also on GPUs can be managed by PSBLAS. However main kernels are the application of the smoother and of the coarsest solver, which represent the key issues to get a good trade-off between convergence properties and parallel performance. Many times, using embarassingly parallel Jacobi smoother can be the solution for smoothing, but direct coarsest solver and more sophysticated smoothers often require the solution of sparse triangular systems that is very inefficient in a parallel environment due to the very low parallel degree and their sequential nature.

This was main motivation of extension of MLD2P4 with a plugin for the computation of sparse approximate inverses. Approximate inverse techniques rely on the assumption that for a given sparse matrix it is possible to find a sparse matrix which is a good approximation of its inverse, so that its application revolves around a sparse-matrix vector product. Currently, CNR and University of Rome “Tor-Vergata” are experimenting approximate inverses as smoother and coarsest solver in the application phase of a multigrid cycle, by exploiting PSBLAS GPU plugin, in order to move preconditioners of MLD2P4 towards GPU accelerators and hybrid distributed/shared memory parallel environments. First promising results of weak scalability on 128 GPUs of the Jureca system operated by JSC are obtained for some linear systems up to 256 millions of unknowns arising from a mini-app emulating Parflow simulations from WP4.

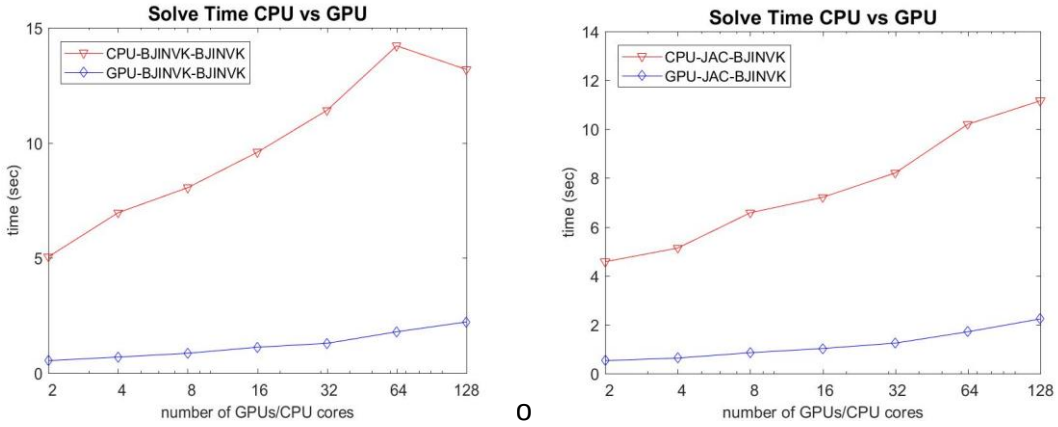


Figure 4. Comparison of matrix solvers on CPU and GPU

Acknowledgement

In case of PSNC the scientific/academic work is co-financed from financial resources for science in the years 2016-2018 granted for the realization of the international project financed by Polish Ministry of Science and Higher Education (agreement number 3543/H2020/2016/2)

5. Developing generalized performance characterization metrics

This activity included the initial design of benchmark tools for the joint EoCoE-POP workshops, regular review of the chosen metrics, as well as deeper follow-up audits of some of the EoCoE applications carried out by POP (see the ParFlow example described below). The initial set of 28 metrics created in consultation with colleagues from the POP CoE are shown below, and are described in some detail in D1.16 (see Section 4 in particular). In practice probably around half of these were measured and monitored by the code teams.

		Metric name	Definition	Tool
Global	1	Total Time (s)	Total application wall time	*time*
	2	Time IO (s)	Average time spent in doing IO for each process	Darshan
	3	Time MPI (s)	Average time spent in MPI for each process	Scalasca
	4	Memory vs Compute Bound	1.0 means strongly compute bound, 2.0 means strongly memory bound	cf text
	5	Load Imbalance	Ratio of the load imbalance overhead towards the critical path duration	Scalasca
IO	1	IO Volume (MB)	Total amount of data read and written	Darshan
	2	Calls (nb)	Total number of IO calls	Darshan
	3	Throughput (MB/s)	IO.1 / Global.2	Computed
	4	Individual IO Access (kB)	IO.1 / IO.2	Computed
MPI	1	P2P Calls (nb)	Average number of peer to peer communications per MPI rank	Scalasca
	2	P2P Calls (s)	Average time spent in peer to peer communications per MPI rank	Scalasca
	3	P2P Message Size (kB)	Average message size in peer to peer communications per MPI rank	Scalasca
	4	Collective Calls (nb)	Average number of collective communications per MPI rank	Scalasca
	5	Collective Calls (s)	Average time spent in collective communications per MPI rank	Scalasca
	6	Collective Message Size (kB)	Average message size in collective communications per MPI rank	Scalasca
	7	Synchro / Wait MPI (s)	Average time spent in synchronization per MPI rank	Scalasca
	8	Ratio Synchro / Wait MPI	MPI.7 / Global.3	Computed
Node	1	Time OpenMP (s)	Time spent in OpenMP parallel region	Scalasca
	2	Ratio OpenMP	Ratio of the time spent in OpenMP parallel region towards the total calculation time	Scalasca
	3	Time Synchro / Wait OpenMP	Average time spent in synchronization/OpenMP overhead per thread	Scalasca
	4	Ratio Synchro / Wait OpenMP	Node.4 / Node.1	Computed
Mem	1	Memory Footprint	Average memory footprint of an MPI process	IdrMem/ Slurm
	2	Cache Usage Intensity	Cache Hit / (Cache Hit + miss) in Last Level Cache	PAPI
Core	1	IPC	Total number of instructions executed / Total number of cycles	PAPI
	2	Runtime without vectorization	Total application wall time compiled with vectorization disabled	*time*
	3	Vectorisation efficiency	Global.1 / Core.2	Computed
	4	Runtime without FMA	Total application wall time when compiled with FMA disabled	*time*
	5	FMA efficiency	Global.1 / Core.4	Computed

5.1. ParFlow follow-up POP analysis - case study

The main goal of the follow-up POP analysis (POP_PP_07) was to investigate ParFlow's (version 693) behavior on Intel Xeon Phi and discover potential fields for improvement as the Intel Xeon Phi processor is considered to be a potential candidate for the future production runs. Most attention was given to two aspects: possible compiler flags which could influence the performance of ParFlow, and vectorization possibilities. Comparison of the runtime on Intel Xeon Phi versus Intel Xeon versus IBM PowerPC A2 was also covered in the POP analysis.

Analysis was dedicated to ParFlow single core performance. ParFlow's solver runtime executed with various compiler flags (e.g. no-simd, align, f-unroll-loops, no-prec-div, ipo etc) as well as profiler guided optimization was compared with runtime of the solver with default

compiler flags. It was revealed that only a few compiler flags (i.e. -no-prec-div, -parallel and profiler guided optimization) can slightly improve application runtime. Moreover, enabled vectorization can slightly improve the runtime of the Solver.

Initial vectorization analysis was performed with Intel Vectorization Advisor. It revealed that in the original version of ParFlow only 0.9% of the total time spent in 10 vectorized loops whereas 99.1% is a scalar code. Efficiency of 10 vectorized loops is 44%. The top consuming loops occur in three functions, i.e. RichardsJacobianEval and PhaseRelPerm and Saturation. Loops in these functions are potential candidates for vectorization optimization. All aforementioned loops use the same macro, i.e. GrGeomInLoop defined in grgeometry.h. The simplified structure after preprocessing the macro constitutes four nested loops, i.e. one outermost while loop which contains three for loops.

In order to improve vectorization, the compiler report recommended to apply loop interchange to the aforementioned loops. In POP analysis all six possibilities were discerned (e.g. ijk, jki etc). However, the measurements show almost no difference in runtime (time varies from 86 to 89s). As the automatic vectorization did not work effectively, it is possible to use a different approach, i.e to guide the compiler to vectorize loops with the #pragma ivdep or combination of #pragma vector always and #pragma ivdep or #pragma simd and try all six possibilities of loop interchange. #pragma simd forces the compiler to vectorize the code whether it is beneficial or not, does not check for aliasing or dependencies that might cause incorrect results, poor performance, memory errors. Whereas #pragma ivdep overrides potential dependencies, but compiler will do dependency analysis and will not be vectorized if it find dependency, and #pragma vector always overrides efficiency heuristics that estimate whether vectorization of a loop is likely to yield a performance benefit.

To investigate how ParFlow behaves on Intel Xeon versus Intel Xeon Phi 7 versus IBM PowerPC A2 measurements of original version of ParFlow on JULIA (Intel Xeon Phi 7210,1.3 GHz) and JURECA (Intel Xeon E5-2680, 2.5 GHz) and JUQUEEN (IBM PowerPC A2, 1.6 GHz) were compared. The same Intel/2017.1.132 compilers were used on JULIA and JURECA and default compiler on JUQUEEN (IBM XL V12.1). Results reveal that ParFlow on JURECA is much faster than on JULIA or JUQUEEN, e.g. solver is 7 times slower on JULIA and 8 times slower on JUQUEEN. It is not a surprise that ParFlow, which has a significant amount of branches (details provided in POP_AR_17), is much slower on JUQUEEN due to its in-order instruction execution. Whereas both KNL (JULIA) and Haswell (JURECA) have out-of-order instruction execution and the rough estimation of the runtime is proportional to processor frequency.

6. Requirements of hardware architecture and programming models

EoCoE contributed to the Strategic Research Agenda (SRA3),[15] compiled and published in December 2017 by the EXDCI project. As part of this exercise, EoCoE developer teams in charge of the most promising applications identified within each domain pillar were asked to assess the challenges remaining in order to push these codes to exascale. It is worth recalling that the goal of the energy CoE is to improve means of production, storage and distribution of clean electricity. This involves areas as diverse as meteorology, where very short term forecasting is needed to predict the production of solar and wind farm and their efficient coupling to the grid and energy trading; fusion for energy, where coupling kinetic and fluid codes is necessary to model the entire chain of processes from vessel core to edge; discovery and design of new energy materials for photovoltaic cells, batteries and supercapacitors; and energy hydrology to manage geothermal and hydro-power including the influence of climate change on these resources. Key applications identified as having a high potential to exploit exascale and which would benefit from reengineering efforts include Gysela (fusion), Parflow (Water), Alya (Meteo), and Metalwalls, BigDFT, and PVegf (Materials).

Exascale computing will enable significant step changes in the predictability and management of renewables as their share of the energy mix increases towards 100% over the coming decades. This translates to a set of specific challenges arising in each domain for wind, solar, hydro and fusion power, as well as energy materials. For example, a single large eddy simulation of turbulent flow through 100 turbines of an entire GW-scale onshore wind farm with complex terrain geometries would require billions of grid points and millions of time steps; the whole thing then repeated for a series of meteorological conditions to obtain an overall power output estimate. Similarly, accurate hydropower prediction relies on the combination of physically-based terrestrial water-for-energy models with observations providing the current state of the hydrologic states and fluxes. Resolving the plasma turbulence that governs the performance of a nuclear fusion reactor from electron scale (~10-4m) up to ITER size (~1m) with realistic time steps (~10-7s) over an energy confinement time (~1s) requires exascale. Low carbon energy technologies cover, among others, energy performance of buildings, harvesting of renewable energy, energy storage and decarbonization. Advanced materials play a vital role in cost reduction, increase in performance and extension of lifetime of these technologies, the design of which often needs to take into account atomic-scale chemistry and how it affects the physical properties at larger device scales. These challenges are particularly relevant to energy technologies such as photovoltaics, batteries and supercapacitors.

A selection of these challenges and their associated algorithmic/machine requirements are given in the table below:

	Computational challenge	**Algorithmic or hardware requirements**

[15] www.etp4hpc.eu/sra

C1	Perform ultra-large O(100-1000) ensemble calculations to generate probabilistic power forecasts	Efficient coupling of capacity and throughput computing, avoiding excessive IO
C2	Perform multiscale LES modelling of entire wind farms with complex terrains	Permit simulations with 10e10-10e12 grid points on unstructured grids with O(1 day) time-to-solution
C3	Perform continental-scale hydrological simulations with mixture of active and inactive regions	Adaptive domain decomposition to mitigate load imbalance
C4	Meteo or hydro modelling using combined elliptic/parabolic equation systems (eg: incompressible Navier-Stokes equations for wind turbines, gyrokinetic or full Maxwell equations for fusion application)	Scalable algebraic solvers (eg multigrid), capable of exploiting accelerator hardware
C5	Avoidance of excessive I/O in ensemble or multi-parameter calculations	Big data handling with fast analytics
C6	Global, self-consistent tokamak modelling including core and wall plasma regions	Efficient statistical coupling of kinetic and fluid models while maintaining high scalability

7. Development of performance models

Following an invited tutorial at the F2F meeting in Toulouse by Gerhard Wellein from the Performance Engineering Group[16], Erlangen University, it was decided to hold a 3-day hackathon on performance engineering for EoCoE applications. This will be organised in partnership with PRACE PATC and the university of Erlangen, and will take place at Maison de la Simulation from June 11th to 13th 2018. The provisional agenda is:

- 1.5 days: lectures on performance engineering methodology
- 1.5 days hands-on: apply the methodology on intensive computing kernels you bring from your code.

8. Evaluating performance and designing balance for different future architectures

Many EoCoE applications also play a part in ongoing FET-HPC projects, where experimental architectures and programming models are made available to developers for trying out prior to potential adoption in future production systems.

16 https://hpc.fau.de/

Project name	Project goals	Opportunities and challenges for EoCoE applications
ALLScale	Develop an exascale programming environment including a unified API to express parallelism at a high level of abstraction	Possible programming model for C++-based EoCoE applications
ANTAREX	Provide a framework that allows to express application self-adaptivity at design-time and to runtime manage and autotune applications for green and Heterogeneous HPC systems up to the Exascale level	Facilitate autotuning for EoCoE applications. The use of a Domain Specific Language approach requires, however, significant code modifications.
ComPat	Develop generic and reusable High Performance Multiscale Computing algorithms that will address the exascale challenges posed by heterogeneous architectures	Exploit expertise created for multiscale materials science as well as use tools for multiscale simulations in general
DEEP-EST	Creation of a modular supercomputing concept to support applications with different computational characteristics and complex workflows	Explore concepts of modular supercomputing for CoE applications and analyse benefits on prototypes created by the project
ECOSCALE	Realisation of a new approach to reconfigurable computing using FPGAs based on the UNILOGIC architecture	Acceleration of application kernels using FPGAs. Development environment possibly in a too early state for complex EoCoE applications plus need for significant porting efforts.
ESCAPE	Develop extreme-scale computing capabilities for European operational numerical weather prediction and future climate models	Project addresses application area that is complementary to parts of EoCoE
Euroexa	Development of an HPC architecture based on ARM processor cores and FPGA accelerators	New architecture for EoCoE applications. Programming model is yet not announced.
ExaFLOW	Address key algorithmic challenges in CFD (Computational Fluid Dynamics) to enable simulation at exascale	Exploit scalable algorithms for Computational Fluid Dynamics calculations
ExaHyPe	Development of new mathematical and algorithmic approaches to solve partial differential equations (PDEs) at exascale	Exploit the open-source software package Hyperbolic PDE Engine

ExaNest	Develop new network, storage and system packaging technologies for future HPC systems	Explore new network and storage architectures for EoCoE applications on the ExaNeSt prototype. As this is based on FPGAs with rather weak ARM cores, porting efforts might become too high.
ExaNoDe	Develop a processor-level packaging technology for a processor design supporting the UNIMEM architecture	Explore new memory architecture for EoCoE applications on the ExaNoDe prototype. As this is based on FPGAs with rather weak ARM cores, porting efforts might become too high.
ExCAPE	Development of scalable machine learning algorithms for complex models and extreme data volumes	While machine learning approaches do play a role in EoCoE, the scalability needs are moderate as of today.
EXTRA	Create a new and flexible exploration platform for developing reconfigurable architectures, design tools and HPC applications with run-time reconfiguration built-in from the start	Acceleration of application kernels using FPGAs. Development environment possibly in a too early state for complex EoCoE applications plus need for significant porting efforts.
greenFLASH	Prototype for a Real-Time Controller targeting the European Extremely Large Telescope	No obvious synergies between greenFLASH and EoCoE
INTERTWInE	Development of application programming interfaces to facilitate efficient and portable use of future exascale architectures with focus on interoperability between different programming models	Enhanced flexibility in exploiting parallel programming models for EoCoE applications as well as optimised versions of applications like iPIC3d and LUDWIG that could be of relevance for energy-oriented research
MANGO	Development of a prototype system for rapid and efficient exploration of manycore architectures for HPC systems with focus on applications demand some form of time-predictability	Application focus of MANGO has little overlap with the application portfolio of EoCoE
MB2020	Initiate the development of a future low-power European processor for Exascale	Opportunity to provision requirements for future processor architectures
MontBlanc-3	Development of a ARM-based hardware	Use of prototype for

	architecture	obtaining experience with ARM-based architectures for EoCoE applications
NextGenIO	Development of a new HPC and HPDA architecture that integrates a byte-addressable storage class memory	Address I/O and metadata handling bottlenecks identified for larger ensemble simulations
NLAFET	Improved exploitation of hardware capabilities by development of novel parallel algorithms, exploration of advanced scheduling strategies and runtime systems, offline and online autotuning, as well as avoiding communication and synchronization bottlenecks	Exploitation of improved numerical libraries released by NLAFET project
READEX	Development of a run-time for automatic tuning for energy-efficiency	Using run-time once available for reducing energy-to-solution for EoCoE applications
SAGE	Development of a scalable, multi-tier, object-store based storage architecture for HPC	Address I/O and metadata handling bottlenecks identified for larger ensemble simulations