Adapting a Finite-Element Type Solver for Bioelectromagnetics to the DEEP-ER Platform

Raphaël Léger, Damian Alvarez Mallon, Alejandro Duran, Stéphane Lanteri

July 2015

Adapting a Finite-Element Type Solver for Bioelectromagnetics to the DEEP-ER Platform

Raphaël LÉGER [a,1], Damian A. MALLÓN [b], Alejandro DURAN [c] and Stéphane LANTERI [a]

[a] *INRIA Sophia-Antipolis Méditerranée, France*
[b] *Jülich Supercomputing Centre, Germany*
[c] *Intel Corporation Iberia, Spain*

Abstract. In this paper, we report on our recent efforts towards adapting a Discontinuous Galerkin Time-Domain solver for computational bioelectromagnetics to the novel, heterogeneous architecture proposed in the DEEP-ER european project on exascale computing. This architecture is based on the Cluster/Booster division concept which will be recalled. As a first step, we summarize the key features of the application and present the outcomes of a profiling of the code using the tools developed by DEEP-ER partners. We then go through the subsequent general improvements of the application as well as specific developments aimed at exploiting efficiently the DEEP-ER platform. This particularly includes porting the application to the Intel® Many Integrated Core Architecture. We conclude with an outlook on next steps, including the different Cluster/Booster division strategies.

Keywords. discontinuous galerkin time-domain, heterogeneous computing, exascale computing, Intel xeon phi, numerical dosimetry

Introduction

Exascale computing will inevitably rely on manycore technology. In the current top 10 systems of the top 500 list, traditionally dominated by proprietary Massively Parallel Processing (MPP) systems, 40% use coprocessors/accelerators. These highly parallel coprocessors give new opportunities to application developers to speed up their simulations. However, they come at a price, as programming them requires extra effort to deal with massive amounts of threads, wide vector units, new levels in the memory hierarchy, and the need of user managed memory transfers.

Intel® Xeon Phi™ products aim at improving the management of this complexity by using a processor architecture which is more familiar to application developers than GPGPU cards are. This architecture enables applications to run in different modes: they can be programmed using the *offload mode* – where the application runs in the normal processors and offloads specific kernels to be executed in the coprocessors – the so called

[1]Corresponding Author: Raphaël Léger, INRIA Sophia-Antipolis Méditerranée, 2004 route des Lucioles, 06902 Sophia-Antipolis CEDEX, France; E-mail: raphael.leger@inria.fr.

native mode – using exclusively the Intel Xeon Phi coprocessors without normal processors being part of the application execution – and *symmetric mode* – where an application can use simultaneously both the processors and coprocessors using MPI communication between them. The next generation of Intel Xeon Phi products will also be available as stand-alone processors. It is worth noting that the Intel®Many Integrated Core Architecture (Intel MIC Architecture) facilitates the programmers job, but a manual effort to improve the threading parallelization, the use of the memory hierarchy or the efficient exploitation of the vector units is generally required to significantly improve the delivered performance.

Exascale requires to expose as much parallelism as possible, and a wise use of resources to keep the power budget under constraints while improving the performance of the applications. To this aim, the DEEP project [1] leverages the first generation of Intel Xeon Phi coprocessor capabilities to decouple the coprocessors from the Cluster nodes (which are otherwise tied to each other, imposing some constraints). The set of coprocessor nodes constitute the so called Booster nodes, which is a separate part of the system with a different fabric enabling autonomous operation of the coprocessors. This allows a dynamic combination of Cluster nodes and Booster nodes in every job. This way, applications can use the coprocessors in two different ways: (1) as dynamically allocated discrete offload destinations, allowing to overcome load imbalance issues, since Cluster nodes with extra work can use more Booster nodes if necessary; and (2) as a set of coprocessors with direct communication, separating different parts of the application according to their scalability. Thus, low scalable parts run on the Cluster, and highly scalable parts – with communication among them – run on the Booster. Following a co-design effort, the OmpSs programming model [2] has been extended to enable both use models.

The DEEP-ER project addresses two issues that were not addressed by its older sibling: scalable I/O and resiliency. Using the DEEP project as foundation, the DEEP-ER project will use the more capable second generation Intel Xeon Phi processor (code-named Knights Landing), and will integrate Booster nodes and Cluster nodes in the same fabric. Together with the Intel Xeon Phi processors, the Booster nodes will include Non Volatile Memory (NVM), to enable very high bandwidth for I/O intensive applications. Taking into account applications requirements, the BeeGFS filesystem [3,4] is being extended to make an efficient use of NVM, where data in the global filesystem will use NVM as cache, prefetching and flushing data as necessary. Additionally, SIONlib [5] is used in the project to alleviate the problem of "small I/O", where hundreds of thousands of small files are written simultaneously by each process in a large job. It does that by aggregating all these files in a reduced set of files, putting less pressure on the metadata servers. Exascale 10 (E10) [6] libraries are also being developed to mitigate this problem when using the MPI-IO interfaces. Efficient I/O enable efficient resiliency. In DEEP-ER resiliency is implemented in two different ways: Scalable Checkpoint-Restart (SCR), and task-based resiliency. The SCR API is being extended to support a number of features, namely "buddy checkpointing" where most of the checkpoints are written to a neighbors' NVM, instead of the global filesystem, allowing an even more scalable approach. The task-based resiliency approach restarts a given OmpSs task using its saved input data, when a transient error has being detected in a memory page that contained data used by the tasks. Moreover, Network Attached Memory (NAM) will be an experimental part of the project, being used to assist in book keeping for I/O and resiliency libraries. An overview of the DEEP-ER hardware architecture is sketched on Figure 1. In a co-design

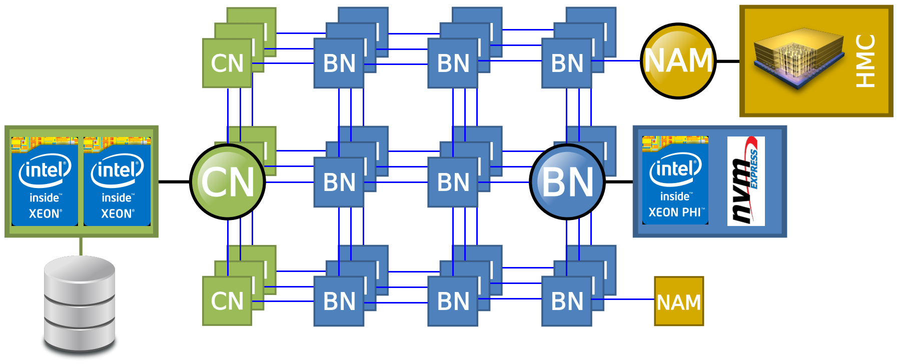


Figure 1. DEEP-ER hardware architecture sketch. The distribution of Cluster Nodes, Booster Nodes and NAM is not yet decided. Cluster nodes and the parallel filesystem storage nodes are merged in the prototype.

effort, the DEEP-ER project involves the adaptation of applications from different areas of science and engineering to its architecture, leveraging its software stack. In this context, the objective of this paper is to give an overview of the steps taken so far in approaching the DEEP-ER hardware and software environment with the application we describe in the following sections.

1. Application Background

In this work, we consider a simulation tool for the numerical assessment of the exposure of human tissues to electromagnetic waves. MAXW-DGTD solves the 3D Maxwell-Debye PDE system [7] in which the unknowns are an electric field **E**, a magnetic field **H** and an electric polarization **P**. The solver is based on a Discontinuous Galerkin Time-Domain (DGTD) method [8], which is a form of finite-element method. It is based on a cellwise local variational formulation and a weak coupling of a given cell to its direct neighbours through a numerical flux (like finite-volume methods). We use a centered numerical flux and nodal Lagrange polynomials basis functions of order k [9]. The family of k^{th} order polynomials is denoted P_k throughout the paper. The value of k impacts the accuracy of the numerical solution through the local amount of discretization points per cell N_k. Lagrange basis functions P_k are such that $N_k = (k+1)(k+2)(k+3)/6$. Equipped with a second order explicit leap-frog time stepping method, the fully discretized scheme for each cell i reads:

$$\left\{\begin{aligned}
\frac{\mathbb{M}_i^{\mu}}{\Delta t}\left(\overline{\mathbf{H}}_i^{n+\frac{3}{2}} - \overline{\mathbf{H}}_i^{n+\frac{1}{2}}\right) &= \sum_{k=1}^{3} \mathbb{K}_i^{x_p} \overline{\mathbf{E}}_i^{n+1} + \sum_{a_{ij}\in\mathscr{T}_d^i} \mathbb{F}_{ij} \overline{\mathbf{E}}_j^{n+1}, \\
\frac{\mathbb{M}_i^{\varepsilon}}{\Delta t}\left(\overline{\mathbf{E}}_i^{n+1} - \overline{\mathbf{E}}_i^{n}\right) &= -\sum_{k=1}^{3} \mathbb{K}_i^{x_p} \overline{\mathbf{H}}_i^{n+\frac{1}{2}} - \sum_{a_{ij}\in\mathscr{T}_d^i} \mathbb{F}_{ij} \overline{\mathbf{H}}_j^{n+\frac{1}{2}} \\
&\quad + \frac{\mathbb{M}_i}{\Delta t}\left(\overline{\mathbf{P}}_i^{n+1} - \overline{\mathbf{P}}_i^{n}\right) + \mathbb{M}_i^{\sigma} \frac{\left(\overline{\mathbf{E}}_i^{n+1} + \overline{\mathbf{E}}_i^{n}\right)}{2}, \\
\frac{2\tau_r^i}{\Delta t}\left(\overline{\mathbf{P}}_i^{n+1} - \overline{\mathbf{P}}_i^{n}\right) &= \varepsilon_0 \left(\varepsilon_s^i - \varepsilon_\infty^i\right)\left(\overline{\mathbf{E}}_i^{n+1} + \overline{\mathbf{E}}_i^{n}\right) - \left(\overline{\mathbf{P}}_i^{n+1} + \overline{\mathbf{P}}_i^{n}\right).
\end{aligned}\right. \tag{1}$$

In (1), $\overline{\mathbf{X}}_i^m$ are the vectors (of size $3 \times N_k$) of discrete unknowns at cell i and time-step m. Cell-local matrices $\mathbb{K}_i^{x_{p=1...3}}$ and $\mathbb{F}_{ij}$ are stiffness and interface (with cell j) matrices. Also, we define $\mathbb{M}_i^{\alpha=\varepsilon,\mu} := \alpha\mathbb{M}_i$ where $\mathbb{M}_i$ are mass matrices. All these matrices are block diagonal, of size $(3 \times N_k)^2$. While the three blocks of mass and stiffness matrices are dense, the blocks of interface matrices are sparse. Finally, n is the time step, and $\varepsilon, \mu, \sigma, \tau_r$ are space-dependent physical input parameters. Spatial discretization of the biological media and surrounding air is based on an unstructured tetrahedral mesh. The traditional approach for exposing parallelism with such a solver is based on mesh partitioning. Each submesh is assigned to a given computing unit, and neighboring submeshes are coupled by exchanging values of unknown fields located at the boundary between subdomains through point-to-point (p2p) MPI messages, once per time step [10]. In [11], Cabel et al. present the adaptation of MAXW-DGTD to a cluster of GPU acceleration cards. For this, they rely on a hybrid coarse-grain SPMD programming model (for inter-GPU communication) / fine-grain SIMD programming model (for intra-GPU parallelization), clearly showing the potential of the DGTD method for leveraging hybrid SIMD/MIMD computing.

2. Analysis and general optimizations

Figure 2 outlines the workflow of the solver and shows the mesh of a human head, together with its partitioning. This mesh is composed of 1.8×10^6 cells for 5 different materials and will be used for all further experiments in this paper. In most simulations, the time loop represents roughly 90% of the total simulation time. While significant I/O activity is mostly concentrated in pre- and post- processing phases - in which we respectively read geometrical data (the mesh and its partitioning), and write the physical

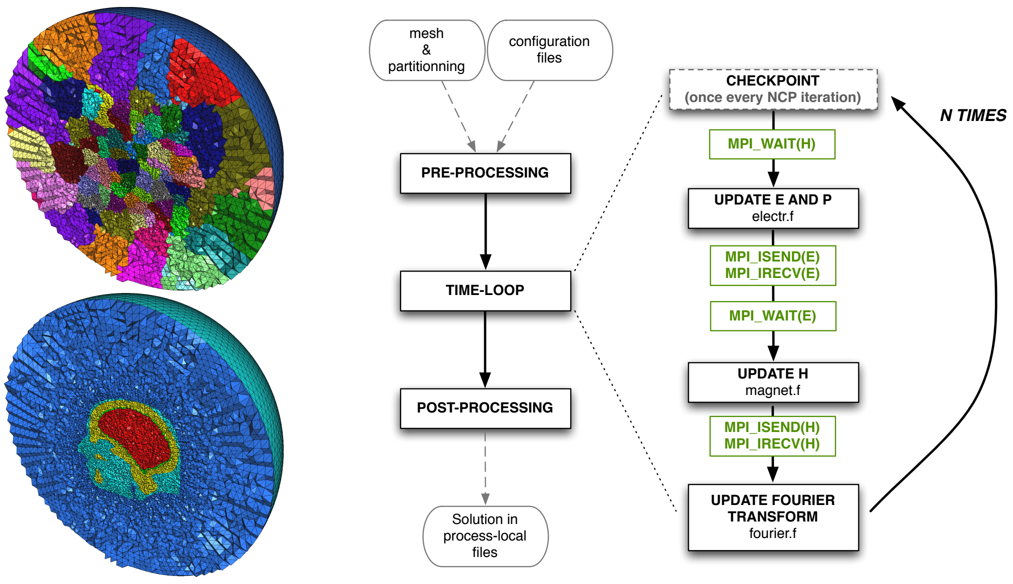

mesh & partitioning
configuration files
PRE-PROCESSING
TIME-LOOP
POST-PROCESSING
Solution in process-local files
CHECKPOINT (once every NCP iteration)
MPI_WAIT(H)
UPDATE E AND P electr.f
MPI_ISEND(E) MPI_IRECV(E)
MPI_WAIT(E)
UPDATE H magnet.f
MPI_ISEND(H) MPI_IRECV(H)
UPDATE FOURIER TRANSFORM fourier.f
N TIMES

Figure 2. Mesh and partitioning - MAXW-DGTD workflow chart

512 proc. (32 nodes)	$k=1$	$k=2$	$k=3$	$k=4$
Parallel eff. - Block / Non-Block	0.56 / 1.01	0.60 / 0.94	0.67 / 0.96	0.76 / 0.97
Speedup due to NB Comms	1.96	1.56	1.44	1.28

Table 1. Performance improvement due to non-blocking communications. Parallel efficiency is computed taking as a reference the walltime observed on 16 processes of the DEEP cluster (i.e. one node).

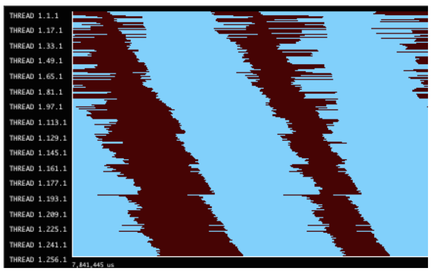
solution in the Fourier space to process-local files - it may also impact the time-loop when application-level checkpointing is activated. The specific topic of I/O is addressed in section 4. Set aside these periodical checkpoints, the time-loop is dominated by the computational operations characterizing the DGTD scheme (1), and, to a lesser extent, constructing on the fly a Fourier-transformed solution. In the rest of this section we rely on trace analysis with Extræ and Paraver [12] of MAXW-DGTD to study and improve communications in 2.1 and data locality in 2.2.

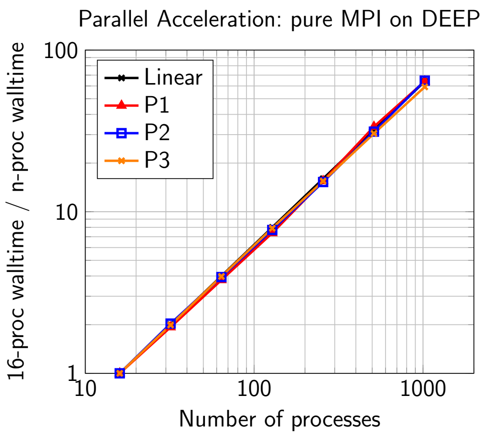
2.1. Processing and communications

In the baseline version of the solver, the approach to p2p communications was based on looping over neighbouring subdomains and relying on blocking MPI_SENDRECV calls in a pipeline fashion. This leads to a pattern represented in figure 5-(a), in which one can see the pipeline approach through the tilted aspect of the MPI events. The late-senders issue which results from this approach has been solved by implementing non-blocking communications allowing to drastically reduce the amount of time spent within MPI routines (partly by enabling the update of the Fourier transform to be overlapped by communication operations for exchanging the values of $\overline{\mathbf{H}}$). Table 1 shows some of the observed performance improvements which result from upgrading communications, allowing to reach very satisfying efficiency values lying close to 1.00. Trace analysis of the baseline version also shows that the portion of runtime spent within communications decreases with the increase of order k. Consequently, upgrading to non-blocking communications leads to a speedup (for a fixed number of MPI ranks) which is higher for lower values of k. Latest strong scaling results for the time loop on the DEEP Cluster[2], based on Intel®Xeon®processor E5-2680, using 16 processes per node up to 64 nodes are represented on figure 5 - (b).

2.2. Improving data locality

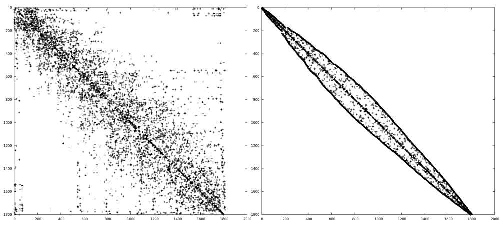
In the application, load balance (which is for an important part a result of the mesh partitioning with the help of the MeTiS [13] tool) is already satisfying with a typical amount of only a few percent of instruction imbalance between processes. This is however not so clear for performance-related metrics such as instructions per cycle (IPC) or typical cache-miss rate. This phenomenon is an effect of an imbalance of data locality. Such an imbalance takes place in the phase which consists in retrieving values of the unknown fields at the neighbouring cells j of a given cell i (i.e. $\overline{\mathbf{E}}_j$ and $\overline{\mathbf{H}}_j$) in order to compute the contribution of the interface-matrix term to the scheme (1). First, one should keep in mind that the $\mathbb{F}_{ij}$ matrices are sparse. Therefore, this phase holds complex gather-scatter access patterns. Second, as the underlying mesh is unstructured, we neither have a priori control on the typical difference of index between one cell and its neighbour (i.e. the




Parallel Acceleration: pure MPI on DEEP
16-proc walltime / n-proc walltime
Number of processes
Linear
P1
P2
P3
100
10
1
10
100
1000

(a) (b)

Figure 3. (a): Paraver visualization of processing events (blue) and MPI_SENDRECV events (brown) in the case of blocking p2p communications involving 256 processes. (b): Scalability results - 20 time loop iterations for a 1.8 million cells human head mesh - pure MPI speedup on the DEEP Cluster using 16 processes per node, from one to 64 nodes.



	No RCM	RCM	Red. factor
(1)	14469	612	23.64 ×
(2)	12216	426	28.68 ×
(3)	5777	293	19.72 ×

(a) (b)

Figure 4. Example of profile of the cell affinity matrix before (a) and after (b) renumbering - Over 128 processes: (1): worst bandwidth, (2): average bandwidth, (3): best bandwidth.

memory locality of the concerned data) nor on its balancing among processes. One way to cope with this is to apply a cell-index renumbering scheme – at the pre-processing stage – aiming to minimizing the bandwidth of the mesh connectivity matrix[3]. For this, we rely on the so-called Reverse Cuthill-McKee [14] algorithm. Figure 4 shows the effect of such renumbering on the connectivity matrix of one subdomain over 128. For this setting and $k = 1...4$, the observed speedups due to RCM renumbering are respectively: 1.26, 1.19, 1.11, 1.05.

3. Approaching the DEEP-ER Booster: porting to the Intel MIC Architecture

In this section, we report on two specific improvements that have been specifically undertaken in order to approach the booster side of the DEEP-ER architecture: threading

[2]Hardware configuration is described in section 6.

[3]We recall the bandwidth of matrix $A = (a_{i,j})$ is the sum of integers $k_{\min} + k_{\max}$ such that $a_{i,j} = 0$ if $j < i - k_{\min}$ or $j > i + k_{\max}$.

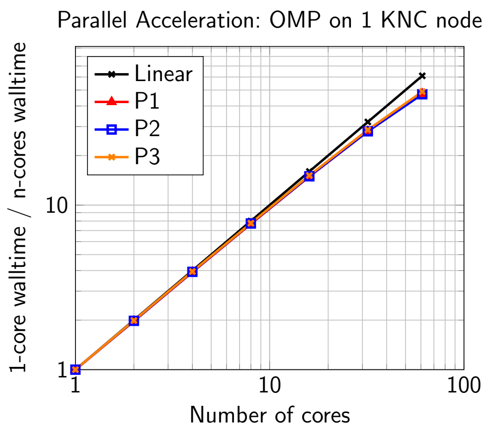


Figure 5. Scalability results - 20 time loop iterations for a 1.8×10^6 cells human head mesh. OpenMP speedup at the level of one Intel Xeon Phi coprocessor, using 4 threads per core, from 1 to 61 cores.

and vectorization.

3.1. Threading

Exploiting efficiently the many-core architecture of the Intel Xeon Phi leads to depart from the initial parallelization approach towards a hybrid MPI/OpenMP programming model. This modification is aiming to minimize the amount of MPI processes required to exploit a given amount of Xeon Phi nodes. The idea is not only in order to limit MPI-overhead but also to alleviate stress on the mesh partitioning phase. An important feature of DGTD methods is the cell-local structure of their formulation, which naturally appears in (1). A direct outcome of (1) is that most of the processing in the application is essentially structured around loops on cells i, which are intrinsically parallel and well load-balanced. Consequently, they lend themselves well to the classical OpenMP parallel loop construct with a static scheduling policy. Applying this strategy leads to an amount of processing left outside any parallel region of the code which represents about only 0.2% of CPU time in the sequential case. NUMA awareness has been implemented by ensuring each thread realizes the first memory access its concerned data. This step is not mandatory to approach the first generation of Intel Xeon Phi coprocessors (codenamed Knights Corner) of the DEEP Booster, but lays the foundation to exploit the NUMA structure of the next generation of Intel Xeon Phi processors (codenamed Knights Landing) which will enable the DEEP-ER Booster. Figure 5 shows strong scaling results of the OpenMP parallelization strategy executed in a node with an Intel Xeon Phi coprocessor. For these experiments, with an affinity policy set to ”compact”, we use 4 threads per core and increase the number of cores from 1 to 61. The observed parallel efficiency for the full 244 available threads on the coprocessor are respectively for $k = 1...4$: 81%, 79%, 82% and 94%.

3.2. Vectorization

Achieving good performance on the Intel MIC architecture requires the ability of efficiently exploiting their 512 bits-wide registers. Using Intel$^{\circledR}$VTune$^{\text{TM}}$ tools to investigate the baseline version of the application (with $k = 3$) using 240 threads of the coprocessor shows that 84% of CPU time in the time-loop is spent in what can be identified as vectorizable kernels. These kernels consist for the most part in applying the cell-local mass, stiffness and interface matrices introduced in (1). A first step towards vectorization

```
DO k=1,ndl
  flx(k) = 0.0d0
  fly(k) = 0.0d0
  flz(k) = 0.0d0
  DO j=1,ndl
    flx(k) = flx(k) + amat(k,j)*flux(1,j,jt)
    fly(k) = fly(k) + amat(k,j)*flux(2,j,jt)
    flz(k) = flz(k) + amat(k,j)*flux(3,j,jt)
  ENDDO
ENDDO
```

OLD

```
DO ic=1,3
  DO k=1,ndl
  temp1=flux(k,ic,jt)
!$OMP SIMD
    DO j=1,ndl
      fl(j,ic) = fl(j,ic) + amat(j,k)*temp1
    ENDDO
  ENDDO
ENDDO
```

NEW

Figure 6. Cell-local Inverted Mass Matrix/Vector product kernel rewriting. Here, jt is the index of the local cell, ndl is N_k, and ic=1...3 describes the 3 blocks of $\mathbb{M}_i^{-1}$ of which coefficients are stored in amat.

improvement consists in upgrading the matrix-vector products to a more efficient syntax, as exemplified in figure 6. This step includes loop reordering and data rearranging (e.g. inverting the two first dimensions of the flux array), as well as relying on OpenMP SIMD constructs to enforce vectorization in innermost loops. A second crucial step is to make sure that concerned data is well-aligned is these rewritten kernels. This is not a priori the case given the values of N_k (renamed ndl in the algorithm of figure 6). The most lightweight solution to this issue is to add padding bytes to the first dimension of concerned double precision arrays in order to reach the next multiple of 64 Bytes, that is to say in the case of amat: amat(1:ndl,1:ndl) becomes amat(1:ndl+MOD(ndl,8),1:ndl). Of course, the same treatment should be applied to intermediate arrays such as the fl array of previous example. It is interesting to remark that first, these modifications come at a very moderate memory cost. The few concerned matrices are indeed shared by all threads of a given process. Second, data alignment almost comes for free on the DEEP cluster nodes based on Intel® microarchitecture code name Sandy Bridge. On these, registers can hold 2 double precision real numbers and N_k happens to be an even number for moderate orders – except for $k = 4$ where $N_k = 35$ and should be padded to 36. Table 2 summarizes the speedup resulting from vectorization enhancement for different values of k, taking the baseline multithreaded version of the code as the reference. One will notice that higher order computations involving larger matrix/vector operations naturally take a better advantage of vectorization optimizations. It is worth noting that the speedup for $k = 4$ is remarkably nearly twice as important as for $k = 1$. Table 3 shows the compared

Speedup with respect to Baseline (OpenMP threading)	$k=1$	$k=2$	$k=3$	$k=4$
Optimized Linear Algebra + OpenMP SIMD + RCM	1.07	1.33	1.32	1.40
... + Padding bytes	1.13	1.63	1.72	2.18

Table 2. Vectorization speedup (full time-loop) - 244 threads on an Intel Xeon Phi coprocessor

performance of the tuned application between the Intel Xeon processors of the DEEP cluster nodes and the Intel Xeon Phi coprocessor for different values of k. For these experiments, the application is run using one process and multiple OpenMP threads on one (out of two) socket of a DEEP cluster node (with 8 cores) and one Booster node using all available 244 threads. It is worth noting that also in that case, the higher the order k is, the greater is the improvement met by the application. In particular, vectorization-oriented optimizations have a decisive impact on the 4th order solver, for which the node to node comparison leans toward the MIC architecture. It is interesting to remark that, first, the larger local problems are, the higher is the reached percentage of peak performance and second, this tendency is more pronounced on the Intel MIC architecture than DEEP cluster nodes.

Intel Xeon Phi coprocessor speedup v.s...	$k = 1$	$k = 2$	$k = 3$	$k = 4$
...DEEP cluster node (one socket, 8 cores)	1.13	1.61	1.55	2.16
... full DEEP cluster node (16 cores)	0.60	0.89	0.87	1.16
%Pperf	$k = 1$	$k = 2$	$k = 3$	$k = 4$
Full Intel Xeon Phi coprocessor (61 cores)	2%	5%	7%	12%
Full DEEP cluster node (16 cores)	14%	21%	32%	38%

Table 3. Performance comparison of the tuned application: Intel Xeon Phi coprocessor speedup with respect to a DEEP cluster node, reached percentage of Peak Performance (%Pperf).

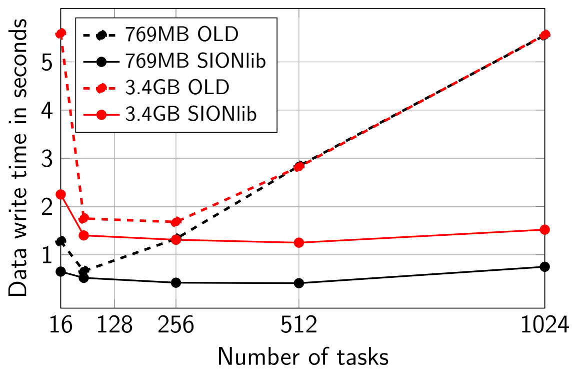


Figure 7. I/O scalability with SIONlib: comparison of outpit times (w/ & w/o SIONlib - DEEP cluster)

4. I/O Scalability

In the application, most of the I/O activity resides in application-level checkpointing and writing the final solution to disk. In both cases the output operations – which involve a total amount of data that is independent from amount of sub-meshes – are performed to process-local files. In the DEEP-ER software stack, SIONlib [5] is an adequate tool to alleviate the stress on the filesystem which occurs when increasing significantly the amount of parallel file access. As such, it has been integrated to the application. Figure 7 shows the improvement in terms of output time due to the use of the SIONlib library, for two different problem sizes corresponding to $k = 1$ and $k = 3$. The number of processes is increased up to 1024 on the DEEP cluster, which we recall is equipped with the BeeGFS filesystem. It is interesting to remark how SIONlib allows to maintain to an almost constant level the output time as the number of processes is increased. This is not the case when the baseline methodology is used. In that situation, the output time grows almost linearly with the number of processes, which is an effect of the filesystem overhead being the vast majority of the time spent in the output routines.

5. Concluding remarks and outlook

The outcome of recent efforts in porting the application to the Intel MIC architecture should allow to reach the best performance out of the DEEP-ER prototype by running at least the time loop in its Booster side based on the next generation of the Intel Xeon Phi processors. The Cluster side might still be leveraged for running some parts of the pre-

and post-processing phases which do not lend themselves well to multithreading, as well as I/O intensive routines. One possibility to achieve this is to consider a model in which these less scalable and I/O phases would be reverse-offloaded from Booster processes to Cluster processes in a one-to-one mapping. This will be achieved by exploiting the OmpSs DEEP offload functionality, developed at Barcelona Supercomputing Center for the DEEP-ER platform. In future work, the OmpSs framework will also be leveraged to expose task-based parallelism and task-based resiliency.

6. Acknowledgments

This work has been funded by the DEEP-ER (Dynamical Exascale Entry Platform - Extended Reach) FP7 project of European Commission (grant agreement no. 610476). The authors would also like to acknowledge the work of the Exascale Cluster Laboratory behind the DEEP-ER project.

Hardware configuration used throughout this work is described in table 4.

	Processor model	Memory	Network	Compiler	OS
(1)	2 x Intel Xeon E5-2680	32GB	Infiniband	Intel15.2.164	CentOS 6.3
(2)	Intel Xeon Phi Coprocessor 7120A	16GB	-	Intel15.1.133	2.6.38.8+MPSS3.2.1

Table 4. Hardware and software configuration - (1) DEEP cluster nodes, (2) Intel Xeon Phi

References

[1] Damian A. Mallon, Norbert Eicker, Maria Elena Innocenti, Giovanni Lapenta, Thomas Lippert, and Estela Suarez. On the scalability of the clusters-booster concept: a critical assessment of the DEEP architecture. In *Proceedings of the Future HPC Systems: The Challenges of Power-Constrained Performance*, FutureHPC '12, pages 3:1–3:10, New York, NY, USA, 2012. ACM.

[2] Alejandro Duran, Eduard Ayguadé, Rosa M Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. OmpSs: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(02):173–193, 2011.

[3] http://www.beegfs.com. BeeGFS/FhGFS, 07 2015. Accessed: 2015-07-09.

[4] Jan Heichler. Picking the right number of targets per server for BeeGFS®. Technical report, ThinkParQ, March 2015.

[5] Wolfgang Frings, Felix Wolf, and Ventsislav Petkov. Scalable massively parallel I/O to task-local files. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 17:1–17:11, Portland, Oregon, 2009.

[6] André Brinkmann, Toni Cortes, Hugo Falter, Julian Kunkel, and Sai Narasimhamurthy. E10 – Exascale IO. White paper, Exascale10 Workgroup, http://www.eiow.org/home/E10-Architecture.pdf, May 2014.

[7] Jonathan Viquerat, Maciej Klemm, Stéphane Lanteri, and Claire Scheid. Theoretical and numerical analysis of local dispersion models coupled to a Discontinuous Galerkin Time-Domain method for Maxwell's equations. Research Report RR-8298, INRIA Sophia-Antipolis Méditerrannée, May 2013.

[8] Loula Fezoui, Stéphane Lanteri, Stéphanie Lohrengel, and Serge Piperno. Convergence and stability of a Discontinuous Galerkin Time-Domain method for the 3D heterogeneous Maxwell equations on unstructured meshes. *ESAIM: Mathematical Modelling and Numerical Analysis*, 39:1149–1176, 11 2005.

[9] Jan S Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin methods: algorithms, analysis, and applications*, volume 54. Springer-Verlag New York Inc, 2008.

[10] Clément Durochat, Stéphane Lanteri, and Raphaël Léger. A non-conforming multi-element DGTD method for the simulation of human exposure to electromagnetic waves. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 27(3):614–625, 2014.

[11] Tristan Cabel, Joseph Charles, and Stéphane Lanteri. Performance evaluation of a multi-gpu enabled finite element method for computational electromagnetics. In Michael Alexander et al., editor, *Euro-Par 2011: Parallel Processing Workshops - CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29 - September 2, 2011, Revised Selected Papers, Part II*, volume 7156 of *Lecture Notes in Computer Science*, pages 355–364. Springer, 2011.

[12] Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: Transputer and occam Developments*, volume 44, pages 17–31. mar, 1995.

[13] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[14] W. Liu and A. Sherman. Comparative analysis of the Cuthill–McKee and the Reverse Cuthill–McKee Ordering Algorithms for Sparse Matrices. *SIAM Journal on Numerical Analysis*, 13(2):198–213, 1976.