Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Exploiting Locality in Sparse Matrix-Matrix Multiplication on the Many Integrated Core Architecture

K. Akbudak[a], C.Aykanat[a]*

[a]*Bilkent University, Computer Engineering Department, 06800 Ankara, Turkey*

Abstract

In this whitepaper, we propose outer-product-parallel and inner-product-parallel sparse matrix-matrix multiplication (SpMM) algorithms for the Xeon Phi architecture. We discuss the trade-offs between these two parallelization schemes for the Xeon Phi architecture. We also propose two hypergraph-partitioning-based matrix partitioning and row/column reordering methods that achieve temporal locality in these two parallelization schemes. Both HP models try to minimize the total number of transfers from/to the memory while maintaining balance on computational loads of threads. The experimental results performed for realistic SpMM instances show that the Intel MIC architecture has the potential for attaining high performance in irregular applications, as well as regular applications. However, intelligent data and computation reordering that considers better utilization of temporal locality should be developed for attaining high performance in irregular applications.

1. Introduction

Sparse matrix-matrix multiplication (SpMM) is an important operation in a wide range of scientific applications such as finite element simulations based on domain decomposition (e.g. finite element tearing and interconnect (FETI) [1]), molecular dynamics (e.g. CP2K [2]), computational fluid dynamics [3], climate simulation [4], and interior point methods [5]. All of these applications exploit parallel processing technology to reduce running times. There exist several software packages that provide SpMM computation for distributed memory architectures such as Trilinos [6], Combinatorial BLAS [7] and for GPUs such as CUSP [8] and CUSPARSE[9].

The objective of this work is to investigate the performance of the Xeon Phi coprocessor for SpMM computations. In order to attain maximum performance on the Xeon Phi architecture, our objective is also to investigate intelligent matrix partitioning and row/column reordering schemes for better utilization of temporal locality to reduce cache misses in the fully-coherent cache implementation of the Xeon Phi architecture.

In this whitepaper, outer-product-parallel and inner-product-parallel SpMM algorithms are proposed and implemented, and the trade-offs between these two parallelization schemes for a single Xeon Phi coprocessor are discussed. Two hypergraph-partitioning (HP) based models and methods that achieve temporal locality in these two parallelization schemes are also proposed and implemented.

* Corresponding author. *E-mail address*: aykanat@cs.bilkent.edu.tr

2. Work done

In the following discussion, we consider the SpMM computation of the form $C = A \times B$, where A and B denote the input matrices, and C denotes the output matrix.

2.1. Outer-product-parallel SpMM

Outer-product-parallel sparse SpMM is based on one-dimensional columnwise and one-dimensional rowwise partitioning of the input matrices A and B as follows:

$$\bar{A} = AP = [A_1^c A_2^c \dots A_K^c] \quad \text{and} \quad \bar{B} = PB = \begin{bmatrix} B_1^r \\ B_2^r \\ \vdots \\ B_K^r \end{bmatrix}$$

Here, K denotes the number of parts and P denotes the permutation matrix obtained from partitioning. The use of the same permutation matrix for column reordering of A and row reordering of B enables conformable columnwise and rowwise partitioning of matrices A and B. In this input partitioning, the SpMM computation is performed as follows:

$$C^k = A_k^c \times B_k^r \quad \text{where} \quad k = 1, \dots, K.$$

Each outer product $C^k = A_k^c \times B_k^r$ will be assigned to a thread that will be executed by an individual core of the Xeon Phi architecture. The nice property of the outer-product formulation is that all nonzeros of the A and B matrices are read only once from the memory. However, multiple writes to the memory will be needed since the output matrix C is computed as follows in terms of the results of thread computations: $C = C^1 + \cdots + C^k + \cdots + C^K$.

We propose and implement an HP-based method to attain temporal locality in the multiple writes to memory during the outer-product-parallel SpMM. In this proposed scheme, input matrices A and B are partitioned recursively and conformably until the size of each output matrix C^k drops below the size of Level 2 cache of the cores of the Xeon Phi architecture. The objective in this partitioning is to allocate A-matrix columns and B-matrix rows that contribute to the same C-matrix entries into the same parts as much as possible. This in turn corresponds to forcing the execution of the outer-product computations that contribute to the same output C-matrix entries on the same core. The proposed hypergraph model encodes this objective successfully thus leading to exploiting the temporal locality in multiple writes to the memory. In the proposed HP model, the partitioning constraint corresponds to maintaining balance on computational loads of threads and the partitioning objective of minimizing cutsize corresponds minimizing the total number of writes to the memory.

2.2. Inner-product-parallel SpMM

Inner-product-parallel sparse SpMM is based on one-dimensional rowwise partitioning of the input matrix A and the output matrix C as follows:

$$\bar{A} = PA = \begin{bmatrix} A_1^r \\ A_2^r \\ \vdots \\ A_K^r \end{bmatrix} \quad \text{and} \quad \bar{C} = PC = \begin{bmatrix} C_1^r \\ C_2^r \\ \vdots \\ C_K^r \end{bmatrix}$$

As in the outer-product parallelization scheme, K denotes the number of parts and P denotes the permutation matrix obtained from partitioning. The use of the same permutation matrix for row reordering of A and row reordering of C shows that the rowwise partition on the input matrix A directly induces a rowwise partition on the output matrix C. In this partitioning scheme, the SpMM computation is performed as follows:

$$C_k^r = A_k^r \times B \quad \text{where} \quad k = 1, \dots, K.$$

Each product $C_k^r = A_k^r \times B$ will be assigned to a thread that will be executed by an individual core of the Xeon Phi architecture. The nice property of the inner-product formulation is that all nonzeros of the A and C matrices are respectively read and written only once. However, multiple reads of rows of the input matrix B will be needed.

We propose and implement an HP-based method to attain temporal locality in the multiple reads from the memory during the inner-product-parallel SpMM computation. In this proposed scheme, input matrix A is partitioned recursively until the storage size of A_k^r together with the required rows of the input matrix B drops below the size of Level 2 cache of the cores of the Xeon Phi architecture. The objective in this partitioning is to allocate A-matrix rows that require the same rows of the B-matrix into the same parts as much as possible. This in turn corresponds to forcing the execution of the SpMM computations that require the same input B-matrix rows on the same core. The proposed hypergraph model encodes this objective successfully thus leading to exploiting the temporal locality in multiple reads. In the proposed HP model, the partitioning constraint corresponds to maintaining balance on computational loads of threads and the partitioning objective of minimizing cutsize corresponds minimizing the total number of reads from the memory.

3. Results

In order to evaluate the validity of the proposed models for outer-product-parallel and inner-product-parallel SpMM algorithms, we compare the performances of our hypergraph-partitioning (HP) based methods to that of the binpacking (BP) method that only considers balancing the computational loads of threads. Here, the BP method is selected as the baseline method for the sake of performance comparison. The BP method uses the best-fit-decreasing heuristic [16]. This heuristic can be effectively used for the solution of the number partitioning problem. So it can be used for load balancing problem. In adapting this heuristic for our purpose, outer/inner products are considered for assignment in decreasing order of their multiply-and-add operation counts. The best-fit criterion is the assignment of the outer/inner products to the minimally loaded bins (parts). The bin capacity constraint is not used in the BP method. Note that number of resulting parts becomes much larger than the number of threads thus enabling the utilization of dynamic part-to-thread scheduling for further load balancing in both methods.

Table 1 displays the properties of the three realistic test matrices used in experimental performance evaluation. The two matrices, *cp2k-h2o-.5e7* and *cp2k-h2o-e6* are obtained from CP2K [2]. These matrices are used for calculating the sign matrix via the Newton-Schulz iterations during water molecule simulations based on Kohn-Sham Density Functional Theory calculations [11]. The *feti-B03* matrix belongs to car engine block simulations based on Finite Element Tearing and Interconnecting (FETI) [12,13] type domain decomposition method. In the table, "SpMM type" shows the type of the SpMM computation utilized in the respective application. In the table, "nrows", "ncols", "nnz", and "sp" denote the number of rows, columns, nonzeros, percent sparsity of the respective test matrices

Table1: Properties of test matrices

Matrix name	**Application**	**SpMM type**	**nrows**	**ncols**	**nnz(A)**	**sp(A)%**	**nnz(C)**	**sp(C)%**
cp2k-h2o-.5e7	Molecular dynamics [2]	AA	279,936	279,936	3,816,315	0.005	17,052,039	0.022
cp2k-h2o-e6	Molecular dynamics [2]	AA	279,936	279,936	2,349,567	0.003	7,846,956	0.010
feti-B03	Finite element [10]	AA^T	6,084	472,320	3,004,692	0.105	258,816	0.699

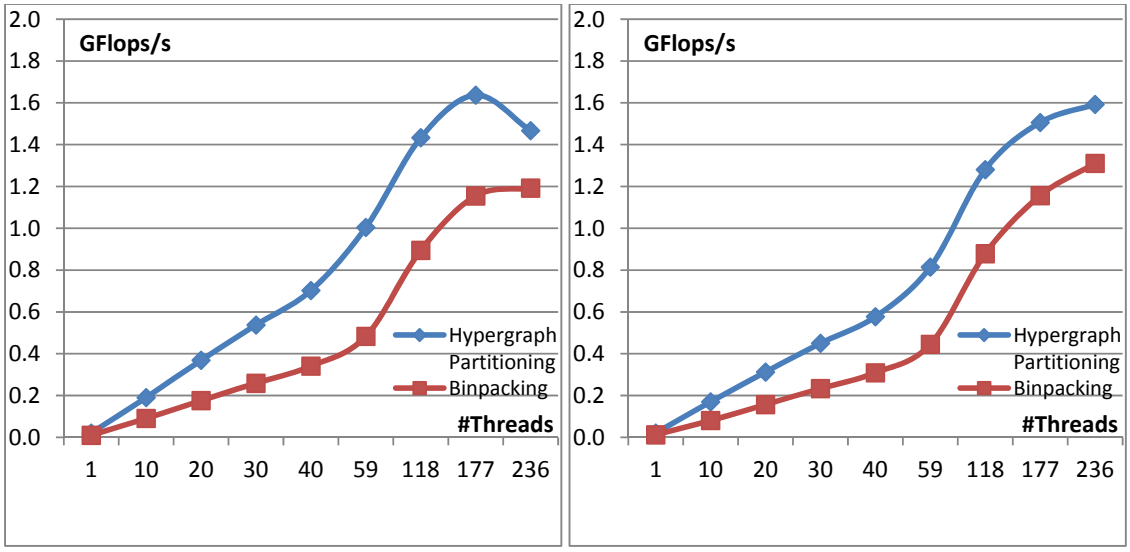
The performance results on a single Xeon Phi P5110 coprocessor are displayed and discussed in the following subsections. We used our local machine which has the same Xeon Phi coprocessor that EURORA has. We used offload mode instead of native mode to enable future vertical integration that involves hybrid parallelization on a

Xeon Phi cluster. The Xeon Phi coprocessor provides 59 cores in offload mode and each core can handle up to four hardware threads. Each core has 512KB Level 2 cache.

3.1. Outer-product-parallel SpMM

Figure 1 shows the performance of the HP and BP methods for the outer-product-parallel SpMM algorithm with increasing number of threads on the Xeon Phi coprocessor. The performance of the outer-product-parallel SpMM algorithm is tested only for the multiplication of the *feti-B03* matrix. This is because, if the output C matrix is stored in compressed sparse format, the sparse accumulation operations required to compute the C matrix from the results of thread computations necessitate complex indexing schemes to efficiently identify the contributions to the same C-matrix entries by different threads. However, the indexing schemes proposed in the literature [7,14] degrade the performance of the outer-product-parallel SpMM computations. Since our main objective was to investigate relative performance of these two parallel schemes, we experimented only with the outer-product-parallel SpMM computation (*feti-B03*) that leads to the smallest C matrix that can fit into Xeon Phi's device memory with dense storage (i.e., zeros are explicitly stored).

As seen in Figure 1, the proposed HP scheme performs considerably better than the baseline scheme BP. In Figures 1(a) and 1(b), full cache size (512KB) and quarter cache size (128KB) are used as thresholds during the recursive bipartitioning operation. Note that the number of resulting parts obtained for the 128KB threshold is approximately four times that for the 512KB threshold. As seen in the figure, the proposed HP scheme attains the maximum performance for three threads per core case, which utilizes 177 threads.

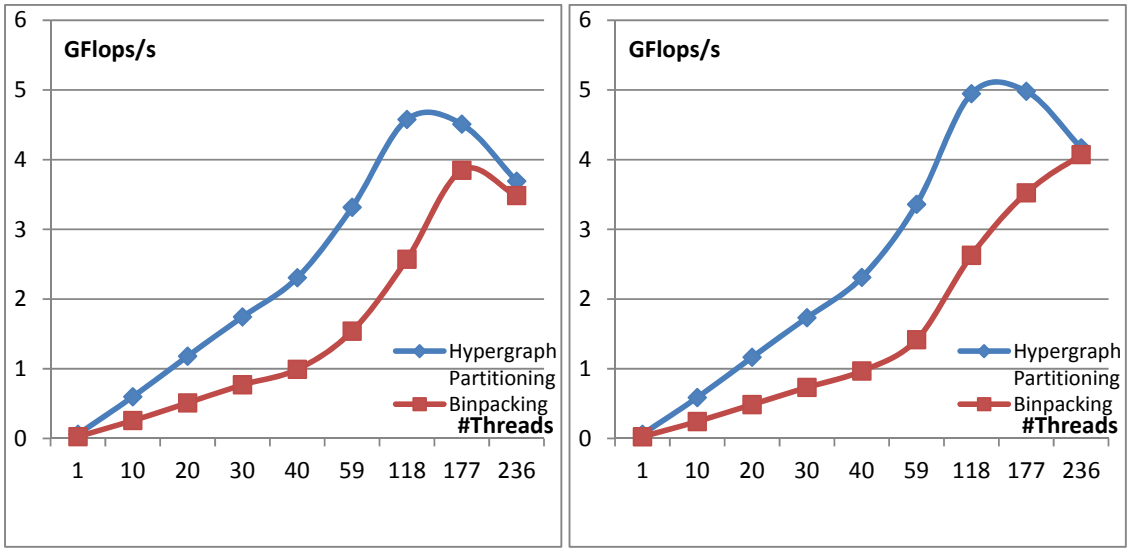


(a) Full cache size threshold (512KB) (b) Quarter cache size threshold (128KB)

Figure 1: Outer-product-parallel SpMM performance for *feti-B03* matrix

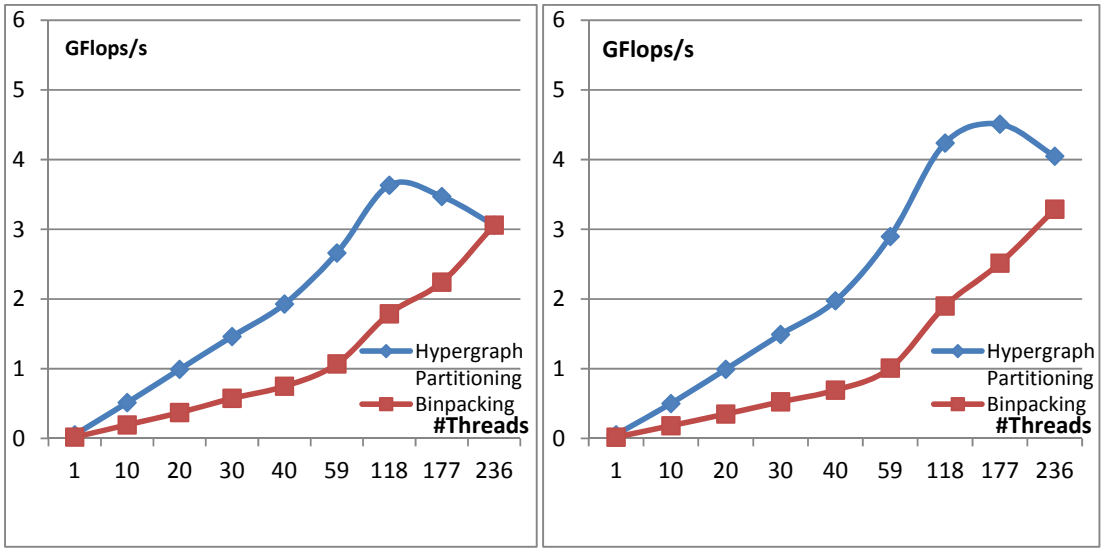
3.2. Inner-product-parallel SpMM

Figures 2, 3, and 4 show the performance of the HP and BP methods for the inner-product-parallel SpMM algorithm with increasing number of threads. As seen in these figures, the proposed HP scheme performs considerably better than the baseline BP scheme. The performance gap between the HP and BP schemes decreases when four threads per core are used but HP still performs slightly better than the BP scheme for this case. As also seen in the figures, the HP scheme attains its peak performance for the three threads per core case except for one-sixteenth cache size threshold of *feti-B03* matrix.



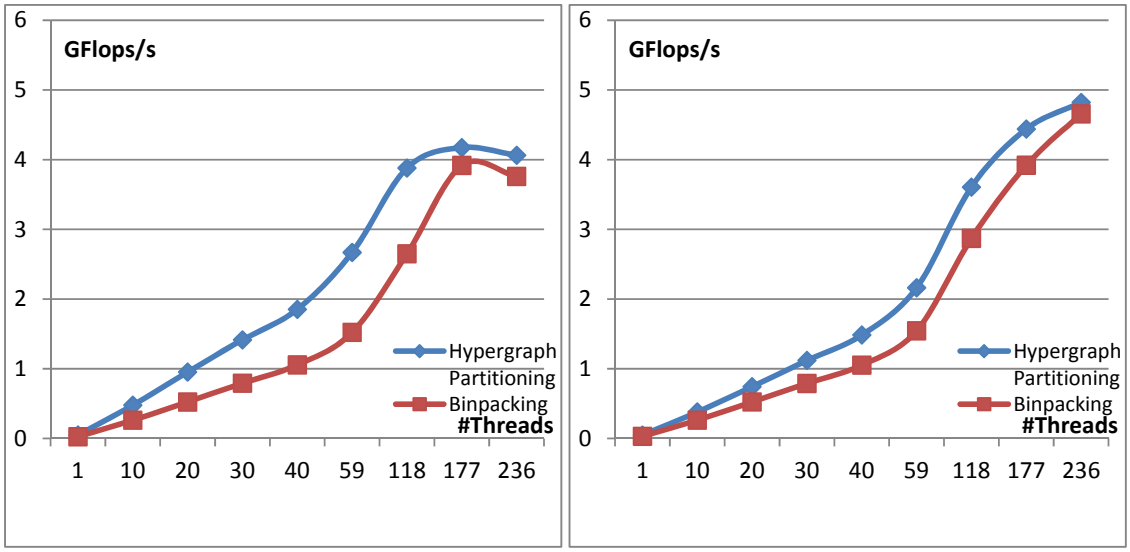
(a) Quarter cache size threshold (128KB) (b) One-sixteenth cache size threshold (32KB)

Figure 2: Inner-product-parallel SpMM performance for *cp2k-h2o-.5e7* matrix



(a) Quarter cache size threshold (128KB) (b) One-sixteenth cache size threshold (32KB)

Figure 3: Inner-product-parallel SpMM performance for *cp2k-h2o-e6* matrix



(a) Quarter cache size threshold (128KB) (b) One-sixteenth cache size threshold (32KB)

Figure 4: Inner-product-parallel SpMM performance for *feti-B03* matrix

3.3. Comparison of outer-product-parallel and inner-product-parallel SpMM algorithms

Comparison of Figures 1 and 4 for the *feti-B03* matrix shows that the inner-product-parallel SpMM algorithm performs significantly better than the outer-product-parallel SpMM algorithm. As seen in these two figures, the inner-product-parallel SpMM algorithm that utilizes the HP scheme attains a peak performance of 4.82 GFlops/s, whereas the peak performance of the outer-product-parallel SpMM algorithm that utilizes the HP scheme is only 1.64 GFlops/s. This experimental finding can be attributed to the locking overhead introduced by the accumulation operations involved in the outer-product-parallel scheme. Depending on these experimental findings, we recommend the use of inner-product-parallel SpMM algorithm.

3.4. Computing power and bandwidth considerations

Here, we present a discussion on the comparison of performance of SpMM computation with respect to the expected peak performance of the Xeon coprocessor in order to provide an insight on the limiting factor of the coprocessor for SpMM computation. Since SpMM computation is not suitable for auto-vectorization provided by the compiler, the peak computing power of the Xeon Phi coprocessor without vectorization is $1.053\ GHz \times 2\ Flops/cycle \times 59\ cores = 124\ GFlops/s$. The peak memory bandwidth of the Xeon Phi coprocessor is $4\ bytes/channel * 16\ channels * 5.0\ GT/s = 320\ GB/s$. Table 2 is introduced to investigate performance bottlenecks of the Xeon Phi coprocessor for SpMM computation. In the table, Flop-to-byte ratio is computed as the total number of multiply and add operations divided by the total number of transfers from/to memory for the respective SpMM instance. The "Max bandwidth" and "Max GFlops/s" columns denote the maximum bandwidth and maximum computing power that can be attained by the Xeon Phi coprocessor depending on the Flop-to-byte ratio.

For the outer-product-parallel SpMM algorithm, maximum bandwidth value attained is 2 GB/s for *feti-B03*, whereas for the inner-product-parallel SpMM algorithm, they are 12, 15, and 9 GB/s for *cp2k-h2o-.5e7*, *cp2k-h2o-e6*, and *feti-B03*, respectively. Comparison of these attained values with the values displayed in the "Max bandwidth" column shows that attained bandwidth values are well below the theoretical bandwidth values. This analysis shows that the memory latency is the bottleneck for not attaining the peak performance rather than the memory bandwidth.

As seen in Figures 1, 2, 3, and 4, the maximum performance of 5 GFlops/s obtained by the SpMM algorithms remains well below the peak computing power of 124 GFlops/s most probably due to the memory latency overhead. As seen in Table 2, the values displayed in the "Max GFlops/s" column are significantly larger (3.35 times larger on the average) than the peak computing power of 124 GFlops/s. This finding shows that parallel SpMM computations will benefit from vectorizing the source code by hand.

Table 2: Theoretical peak performance of the Xeon Phi coprocessor for SpMM computation

	Flop-to-byte ratio	Max bandwidth	Max GFlops/s
cp2k-h2o-.5e7	1.46	85 GB/s	466 GFlops/s
cp2k-h2o-e6	1.41	88 GB/s	451 GFlops/s
feti-B03	1.03	121 GB/s	329 GFlops/s

4. Conclusion

Outer-product-parallel and inner-product-parallel sparse matrix-matrix multiplication (SpMM) algorithms were designed, developed, and implemented for the Xeon Phi architecture. Hypergraph-partitioning-based models and methods are proposed to achieve temporal locality in these two parallelization schemes. Experimental results on realistic datasets showed that the proposed hypergraph-partitioning-based matrix partitioning methods achieve considerable performance improvement over a baseline scheme that only considers balancing the computational loads of threads. The inner-product-parallel SpMM algorithm was found to perform significantly better than the outer-product-parallel SpMM algorithm in our tests. Further work is needed to develop efficient indexing schemes for the sparse accumulation operations required in the outer-product-parallel scheme.

References

[1] Z. Dostál, D. Horák, and R. Kučera; "Total FETI–an easier implementable variant of the FETI method for numerical solution of elliptic PDE", Communications in Numerical Methods in Engineering, 22 (2006), pp. 1155–1162.

[2] CP2K home page. http://www.cp2k.org/.

[3] V. Marion-Poty and W. Lefer; "A wavelet decomposition scheme and compression method for streamline-based vector field visualizations", Computer & Graphics, 26 (2002), pp. 899 – 906.

[4] W. Sawyer and P. Messmer; "Parallel grid manipulations for general circulation models", in Parallel Processing and Applied Mathematics, vol. 2328 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, pp. 605–608.

[5] E. Boman, O. Parekh, and C. Phillips; "LDRD final report on massively-parallel linear programming: the parPCx system", technical report, SAND2004-6440, Sandia National Laboratories, 2005.

[6] M. Heroux, R. Bartlett, V. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, et al.; "An overview of Trilinos", technical report, SAND2003-2927, Sandia National Laboratories, 2003.

[7] A. Buluç and J. R. Gilbert; "Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments", SIAM Journal on Scientific Computing (SISC), 34 (2012), pp. 170 – 191.

[8] N. Bell and M. Garland; "CUSP: Generic parallel algorithms for sparse matrix and graph computations", 2010. Version 0.1.0.

[9] NVIDIA, CUSPARSE library, 2010

[10] T. Kozubek et. al., “FETI Coarse Problem Parallelization Strategies and their Comparison”, Available at: http://www.prace-ri.eu/IMG/pdf/feticoarseproblemparallelization.pdf

[11] J. VandeVondele, U. Borstnik, and J. Hutter, Linear scaling self-consistent field calculations with millions of atoms in the condensed phase, J. Chem. Theory Comput., 8 (2012), pp. 3565–3573.

[12] C. Farhat and F.-X. Roux; “A method of finite element tearing and interconnecting and its parallel solution algorithm”, Int. J. for Numer. Methods Eng., 32 (1991), pp. 1205–1227.

[13] Z. Dostál, D. Horák, and R. Kučera; “Total FETI–an easier implementable variant of the FETI method for numerical solution of elliptic PDE”, Commun. Numer. Methods Eng., 22 (2006), pp. 1155–1162.

[14] A. Buluc and J. Gilbert.; “On the representation and multiplication of hypersparse matrices”, IPDPS, pages 1 – 11, 2008.

[15] http://ark.intel.com/products/71992/Intel-Xeon-Phi-Coprocessor-5110P-8GB-1_053-GHz-60-core

[16] E. Horowitz, S. Sahni; “Fundamentals of Computer Algorithms”, Computer Science Press, Rockville, MD, 1978.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-261557.