Many Core Acceleration of the Boundary Element Method

Michal Merta[1], Jan Zapletal[1(✉)], and Jiri Jaros[2]

[1] IT4Innovations National Supercomputing Center, VŠB-Technical University of Ostrava, 17. listopadu 15/2172, 708 33 Ostrava, Czech Republic
{michal.merta,jan.zapletal}@vsb.cz
[2] Department of Computer Systems, Brno University of Technology, Božetěchova 1/2, 612 66 Brno, Czech Republic
jarosjir@fit.vutbr.cz

Abstract. The paper presents the boundary element method accelerated by the Intel Xeon Phi coprocessors. An overview of the boundary element method for the 3D Laplace equation is given followed by the discretization and its parallelization using OpenMP and the offload features of the Xeon Phi coprocessor are discussed. The results of numerical experiments for both single- and double-layer boundary integral operators are presented. In most cases the accelerated code significantly outperforms the original code running solely on Intel Xeon processors.

Keywords: Boundary element method · Intel Many Integrated Core architecture · Acceleration · OpenMP parallelization

1 Introduction

The necessity to prepare existing scientific codes for the upcoming many-core era has been discussed in numerous works since the introduction of Intel's Many Integrated Core (MIC) architecture in 2010 [3,4]. According to the June edition of 2015 Top 500 list, 35 out of 500 most powerful supercomputers are equipped with the first generation of Intel Xeon Phi coprocessor (Knights Corner, KNC). This number is expected to grow with the introduction of the second (Knights Landing, KNL) and third (Knights Hill, KNH) generations. Announced systems utilizing these architectures include the NERCS's next supercomputing system Cori with the theoretical peak performance of 30 PFLOPS or Argonne's future Aurora supercomputer with the estimated peak performance of up to 450 PFLOPS, which are expected to be operational as of 2016 and 2018, respectively.

The current generation of the Xeon Phi coprocessors features up to 61 cores with four hyper-threads per core and 350 GB/s memory bandwidth. The extended 512-bit wide AVX vector unit provides support for a concurrent SIMD operation on eight double-precision or 16 single-precision operands. The coprocessor offers two main usage modes: the native execution model, when the application directly runs on the coprocessor, and the offload model, when the application runs on the host

T. Kozubek et al. (Eds.): HPCSE 2015, LNCS 9611, pp. 116–125, 2016.
DOI: 10.1007/978-3-319-40361-8_8

processor which offloads certain parts of the code to the coprocessor (see Fig. 1). In this paper we focus on the latter approach. The future KNL and KNH architectures will not only be available in the form of coprocessor cards, but also as stand-alone host processors, which will eradicate the need for data movement via the PCIe interface.

In this paper we demonstrate the performance gain achievable by Intel Xeon Phi acceleration in the field of boundary integral equations. The boundary element method (BEM) is suitable for the solution of partial differential equations which can be formulated in the form of boundary integral equations, i.e., for which the so-called fundamental solution is known. Since BEM reduces the problem to the boundary of the computational domain, it is well suited for problems such as exterior sound scattering or shape optimization. However, the classical method has quadratic computational and memory complexity with respect to the number of surface degrees of freedom and produces fully populated system matrices. In addition, a special quadrature method has to be applied due to the singularities in the kernel of the underlying boundary integrals [12,14]. Although one can employ several fast BEM techniques to reduce the complexity to almost linear (see, e.g., [1,11–13]) an efficient implementation and parallelization of the method is necessary to enable solution of large scale problems. In [9] we have presented an explicit vectorization of BEM, whereas in [7] a new approach based on the graph theory for the system matrix distribution among MPI processes was introduced. In this paper we focus on the acceleration of the classical BEM by the Intel Xeon Phi coprocessors. Similar topic has been discussed, e.g., in [6], where the coprocessors were employed to accelerate the evaluation of the representation formula. We, on the other hand, investigate the assembly of the BEM system matrices necessary to compute the complete Cauchy data of the solution.

The structure of the paper is as follows. In Sect. 2 we introduce the model problem represented by the potential equation in 3D and present its boundary formulation. Section 3 concentrates on the discretization of the boundary integral equations introduced in Sect. 2 and discusses the parallelization approach used in the numerical experiments presented in Sect. 4. We summarize the results and conclude in Sect. 5.

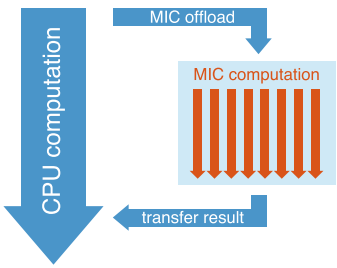

Fig. 1. Using the offload mode of Intel Xeon Phi to accelerate computation.

2 Boundary Integral Equations for the Laplace Equation

Despite its simplicity, the Laplace (or the potential) equation finds its application in numerous fields. Harmonic functions, i.e., the solutions to the Laplace equation, can be used to model the steady-state heat flow or as potentials of electrostatic or gravitational fields.

In the following we consider the mixed boundary value problem for the Laplace equation

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega, \\ u = g & \text{on } \Gamma_\mathrm{D}, \\ \dfrac{\partial u}{\partial \boldsymbol{n}} = h & \text{on } \Gamma_\mathrm{N}. \end{cases} \tag{1}$$

In (1), $\Omega \subset \mathbb{R}^3$ denotes a bounded Lipschitz domain with the boundary composed of two components $\partial\Omega = \overline{\Gamma_\mathrm{D}} \cup \overline{\Gamma_\mathrm{N}}$, $\Gamma_\mathrm{D} \cap \Gamma_\mathrm{N} = \emptyset$, and the boundary data $g \in H^{1/2}(\Gamma_\mathrm{D})$, $h \in H^{-1/2}(\Gamma_\mathrm{N})$. The solution to the Laplace equation is given by the representation formula [8,12,14]

$$\begin{aligned} u(\boldsymbol{x}) = &\int_{\Gamma_\mathrm{D}} \gamma^1 u(\boldsymbol{y})\, v(\boldsymbol{x},\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} + \int_{\Gamma_\mathrm{N}} h(\boldsymbol{y})\, v(\boldsymbol{x},\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} \\ &- \int_{\Gamma_\mathrm{D}} g(\boldsymbol{y})\, \frac{\partial v}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x},\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} - \int_{\Gamma_\mathrm{N}} \gamma^0 u(\boldsymbol{y})\, \frac{\partial v}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x},\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} \quad \text{for } \boldsymbol{x} \in \Omega \end{aligned}$$

with the Dirichlet and Neumann trace operators γ^0 and γ^1, respectively, and the fundamental solution $v\colon \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$,

$$v(\boldsymbol{x},\boldsymbol{y}) := \frac{1}{4\pi} \frac{1}{\|\boldsymbol{x}-\boldsymbol{y}\|}.$$

The unknown Cauchy data $\gamma^0 u|_{\Gamma_\mathrm{N}}$, $\gamma^1 u|_{\Gamma_\mathrm{D}}$ can be obtained from the symmetric system of boundary integral equations [12,15,16]

$$\begin{aligned} (Vs)(\boldsymbol{x}) - (Kt)(\boldsymbol{x}) &= \frac{1}{2}\tilde{g}(\boldsymbol{x}) + (K\tilde{g})(\boldsymbol{x}) - (V\tilde{h})(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma_\mathrm{D}, \\ (K^*s)(\boldsymbol{x}) + (Dt)(\boldsymbol{x}) &= \frac{1}{2}\tilde{h}(\boldsymbol{x}) - (K^*\tilde{h})(\boldsymbol{x}) - (D\tilde{g})(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Gamma_\mathrm{N} \end{aligned} \tag{2}$$

involving the single-layer, double-layer, adjoint double-layer, and hypersingular boundary integral operators

$$\begin{aligned} (Vq)(\boldsymbol{x}) &:= \int_{\partial\Omega} v(\boldsymbol{x},\boldsymbol{y})\, q(\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}, & (Kt)(\boldsymbol{x}) &:= \int_{\partial\Omega} \frac{\partial v}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x},\boldsymbol{y})\, t(\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}, \\ (K^*q)(\boldsymbol{x}) &:= \int_{\partial\Omega} \frac{\partial v}{\partial \boldsymbol{n}_{\boldsymbol{x}}}(\boldsymbol{x},\boldsymbol{y})\, q(\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}, & (Dt)(\boldsymbol{x}) &:= -\gamma^1 \int_{\partial\Omega} \frac{\partial v}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x},\boldsymbol{y})\, t(\boldsymbol{y})\, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}, \end{aligned}$$

respectively. The unknown functions s, t in (2) are defined as

$$\widetilde{H}^{1/2}(\Gamma_\mathrm{N}) \ni t := \gamma^0 u - \tilde{g}, \quad \widetilde{H}^{-1/2}(\Gamma_\mathrm{D}) \ni s := \gamma^1 u - \tilde{h},$$

with suitable extensions $\tilde{g}$, $\tilde{h}$ of the given boundary data g, h.

To compute the unknown functions s, t from (2) we employ the equivalent variational formulation

$$a(s,t,\psi,\varphi) = F(\psi,\varphi) \quad \text{for all } \psi \in \widetilde{H}^{-1/2}(\Gamma_{\mathrm{D}}),\ \varphi \in \widetilde{H}^{1/2}(\Gamma_{\mathrm{N}}) \tag{3}$$

with the bilinear form

$$a(s,t,\psi,\varphi) := \langle Vs,\psi\rangle_{\Gamma_{\mathrm{D}}} - \langle Kt,\psi\rangle_{\Gamma_{\mathrm{D}}} + \langle K^*s,\varphi\rangle_{\Gamma_{\mathrm{N}}} + \langle Dt,\varphi\rangle_{\Gamma_{\mathrm{N}}}$$

and the right-hand side

$$F(\psi,\varphi) := \left\langle \left(\frac{1}{2}I + K\right)\tilde{g},\psi\right\rangle_{\Gamma_{\mathrm{D}}} - \langle V\tilde{h},\psi\rangle_{\Gamma_{\mathrm{D}}} + \left\langle \left(\frac{1}{2}I - K^*\right)\tilde{h},\varphi\right\rangle_{\Gamma_{\mathrm{N}}} - \langle D\tilde{g},\varphi\rangle_{\Gamma_{\mathrm{N}}}.$$

3 Discretization and Parallelization

3.1 Discretization of the Problem

In order to solve the problem (3) numerically, we discretize the boundary $\partial\Omega$ into flat shape-regular triangles τ_i, $i \in \{1,\dots,E\}$. The boundary energy spaces are discretized by continuous piecewise linear functions $(\varphi_j)_{j=1}^N \subset H^{1/2}(\partial\Omega)$ and piecewise constant functions $(\psi_i)_{i=1}^E \subset H^{-1/2}(\partial\Omega)$. Following [12], the discrete system of linear equations reads

$$\begin{bmatrix} \mathsf{V}_h^{\mathrm{DD}} & -\mathsf{K}_h^{\mathrm{DN}} \\ (\mathsf{K}_h^{\mathrm{DN}})^{\mathsf{T}} & \mathsf{D}_h^{\mathrm{NN}} \end{bmatrix} \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{t} \end{bmatrix} = \begin{bmatrix} -\mathsf{V}_h^{\mathrm{DN}} & \frac{1}{2}\mathsf{M}_h^{\mathrm{DD}} + \mathsf{K}_h^{\mathrm{DD}} \\ \frac{1}{2}(\mathsf{M}_h^{\mathrm{NN}})^{\mathsf{T}} - (\mathsf{K}_h^{\mathrm{NN}})^{\mathsf{T}} & -\mathsf{D}_h^{\mathrm{ND}} \end{bmatrix} \begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{g} \end{bmatrix}$$

with $\mathsf{X}^{\bullet\bullet}$ denoting the restrictions of the matrices $\mathsf{X} \in \{\mathsf{V}_h, \mathsf{K}_h, \mathsf{D}_h, \mathsf{M}_h\}$,

$$\mathsf{V}_h[\ell,j] := \int_{\tau_\ell}\int_{\tau_j} v(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{x}}, \qquad \mathsf{M}_h[\ell,i] := \int_{\tau_\ell} \varphi_i(\boldsymbol{x})\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{x}},$$

$$\mathsf{K}_h[\ell,i] := \int_{\tau_\ell}\int_{\partial\Omega} \frac{\partial v}{\partial \boldsymbol{n}_{\boldsymbol{y}}}(\boldsymbol{x},\boldsymbol{y})\,\varphi_i(\boldsymbol{y})\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{x}}, \qquad \mathsf{D}_h := \mathsf{T}^{\mathsf{T}}\mathrm{diag}\{\mathsf{V}_h,\mathsf{V}_h,\mathsf{V}_h\}\mathsf{T}$$

to the respective parts of the boundary. Since M_h is a sparse matrix with no singularity in the integrand and D_h can be computed by a sparse transformation of the single-layer matrix V_h (see [2,12]), in the following we only concentrate on the efficient assembly of the matrices V_h, K_h.

To deal with the singularities in the surface integrals we use the technique proposed in [9,14]. Using a series of substitutions

$$\boldsymbol{F}_n \circ \ldots \circ \boldsymbol{F}_1 =: \boldsymbol{F}\colon [0,1]^4 \to \tau_\ell \times \tau_j,$$
$$\boldsymbol{F}(z_1,z_2,z_3,z_4) = (\boldsymbol{x},\boldsymbol{y}), \quad \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{x}} = \boldsymbol{S}(z_1,z_2,z_3,z_4)\,\mathrm{d}z_1\,\mathrm{d}z_2\,\mathrm{d}z_3\,\mathrm{d}z_4,$$

the matrix entries for V_h (similarly for K_h) read

$$\mathsf{V}_h[\ell,j] = \int_0^1\int_0^1\int_0^1\int_0^1 v(\boldsymbol{F}(z_1,z_2,z_3,z_4))\,\boldsymbol{S}(z_1,z_2,z_3,z_4)\,\mathrm{d}z_1\,\mathrm{d}z_2\,\mathrm{d}z_3\,\mathrm{d}z_4.$$

Since the transformed integrand $h := (v \circ \boldsymbol{F})\boldsymbol{S}$ is analytic, the values can be computed by a standard tensor Gauss integration scheme

$$\sum_{m=1}^{k}\sum_{n=1}^{k}\sum_{o=1}^{k}\sum_{p=1}^{k} \omega_m\,\omega_n\,\omega_o\,\omega_p\,h(z_m, z_n, z_o, z_p). \tag{4}$$

The substitution $\boldsymbol{F}$ is different for τ_ℓ, τ_j being identical, sharing exactly one edge, one vertex, or being disjoint. Note that for well separated triangles, say,

$$\|\boldsymbol{t}_\ell - \boldsymbol{t}_j\| > \eta \max\{\operatorname{diam}\tau_\ell, \operatorname{diam}\tau_j\}$$

with the centres of gravity $\boldsymbol{t}_\ell$, $\boldsymbol{t}_j$ and a suitable coefficient η the kernel function is smooth and the entries can be computed directly by a triangle Gauss integration scheme (see [12], Appendix C)

$$\sum_{m=1}^{k}\sum_{n=1}^{k} \omega_m\,\omega_n\,v(\boldsymbol{x}_m, \boldsymbol{y}_n). \tag{5}$$

3.2 Parallelization and Acceleration of the System Matrix Assembly

The simplified algorithm of the system matrix assembly on a CPU is depicted in Listing 1.1. The algorithm iterates through the couples of elements and assembles local system matrices. The Gaussian quadrature (5) is used for well separated elements; in other cases we employ the scheme (4).

```
1  #pragma omp parallel for
2  for (int i = 0; i < nElements; i++) {
3      for (int j = 0; j < nElements; j++) {
4          if (areElementsDistant(i, j)) {
5              getLocalMatrixFarfield(i, j, localMatrix);
6          } else {
7              getLocalMatrixNearfield(i, j, localMatrix);
8          }
9          globalMatrix.add(i, j, localMatrix);
10     }
11 }
```

Listing 1.1. Simplified CPU computation of the system matrix

After the assembly of the local matrix its contribution is added to the appropriate positions of the global system matrix (line 9 of the listing). Since the computation is done in parallel using OpenMP, the update of the global matrix has to be treated using atomic operations.

To accelerate the computation we distribute the workload among the available accelerators and the CPU by splitting the matrix into $N_{\text{MIC}} + 1$ horizontal blocks. Dimensions of the blocks should follow the ratio between the theoretical peak performance of the Xeon Phi and the host. The computation on the CPU and the coprocessors is performed simultaneously (see Fig. 1). Moreover, to keep the computation on the coprocessors as simple as possible, the accelerators only take care of the quadrature over disjoint elements. The quadrature over close and identical elements are computed on the CPU.

The process consists of several steps:

1. Preallocation of data structures (mainly the system matrix) on the host.
2. Sending the necessary data from the host to the coprocessor (mesh elements, nodes, normals, etc.).
3. Parallel computation on the coprocessor and the host.
4. Sending the partially computed system matrix from the coprocessor to the host.
5. Combination of the host and coprocessor data.

Since the amount of memory on Xeon Phi is usually lower than on the host, the matrix blocks assigned to the coprocessor are further split into chunks of smaller dimensions in order to tackle large problems. Moreover, splitting the matrix enables us to use the technique of double buffering to overlap the data transfer of a matrix chunk with the computation of the subsequent one.

The simplified source code of the offload is provided in Listing 1.2. The underlying data structures are extracted from C++ objects in order to send raw data to the coprocessor using the `#pragma offload target(mic)` statement provided by the Intel Compiler (see lines 1–11 of the listing). The parallelization on the coprocessor is performed using the ordinary OpenMP pragmas. The singular integration is skipped and left for the host. The asynchronous offloaded computation is enabled by the `signal` clause. After the data is sent to the coprocessor the CPU continues in the parallel quadrature over its portion of elements.

```
1  // get pointers to raw data
2  double * nodes  = mesh->getNodes();
3  int * elements = mesh->getElements();
4  double * matrixData = globalMatrix.getData();
5  char s;
6
7  // initialize offload region
8  #pragma offload target(mic) signal(&s) \
9      in(nodes : length(3 * nNodes)) \
10     in(elements : length(3 * nElements)) \
11     out(matrixData : length(dataLength))
12     {
13         #pragma omp parallel for
14         for (int i = myElemStart; i < myElemEnd; i++) {
15             for (int j = 0; j < nElements; j++) {
16                 if (areElementsDistant(i, j)) {
17                     getLocalMatrixFarfieldMIC(i, j, nodes,
       elements,
18                         localMatrix);
19                 } else {
20                     continue;
21                 }
22                 addToGlobalmatrix(i, j, localMatrix,
       matrixData);
23             }
24         }
25     }
26 // simultaneous computation on the host CPU
27 ...
28 // receive data from the coprocessor
29 #pragma offload target(mic:0) wait(&s)
30 // combine the data from the host and the coprocessor
```

Listing 1.2. Simplified offloaded computation of the system matrix

After finishing its work the CPU waits for the data from the coprocessor (see line 29). Finally, the contributions from the host processor and the coprocessor are combined.

Note that the method assembling the local matrix on the Intel Xeon Phi coprocessor has to be appropriately modified. For optimal memory movement the data have to be aligned to 64 byte boundaries. Moreover, the original loops over quadrature points are manually unrolled and vectorization is assisted using the `simd` pragma.

```
1 int out = nOutQPoints;
2 int in = nInQPoints;
3 double value = 0.0;
4 for (int i = 0; i < out; i++) {
5     for (int j = 0; j < in; j++) {
6         value += ...;
7     }
8 }
```

Listing 1.3. Gauss quadrature on CPU

```
1 int p = nOutQPoints*nInQPoints;
2 double value = 0.0;
3 #pragma simd vectorlength(8) \
4     reduction(+:value)
5 for (int i = 0; i < p; i++) {
6     value += ...;
7 }
8
```

Listing 1.4. Gauss quadrature on MIC

Although the compiler automatically vectorizes the inner loop in Listing 1.3, the longer loop in Listing 1.4 is more suitable for the extended SIMD registers on the coprocessor.

4 Numerical Experiments

The following numerical experiments were carried out on the MIC-accelerated nodes of the Salomon cluster at the IT4Innovations National Supercomputing Center, Czech Republic. The nodes are equipped with two 12-core Intel Xeon E5-2680v3 processors, 128 GB of RAM, and two Intel Xeon Phi 7120P coprocessor cards. Each coprocessor offers 61 cores running at 1.238 GHz, four hyperthreads per core, and extended 512-bit vector registers. On the coprocessor 16 GB of memory is available with the maximum memory bandwidth of 352 GB/s.

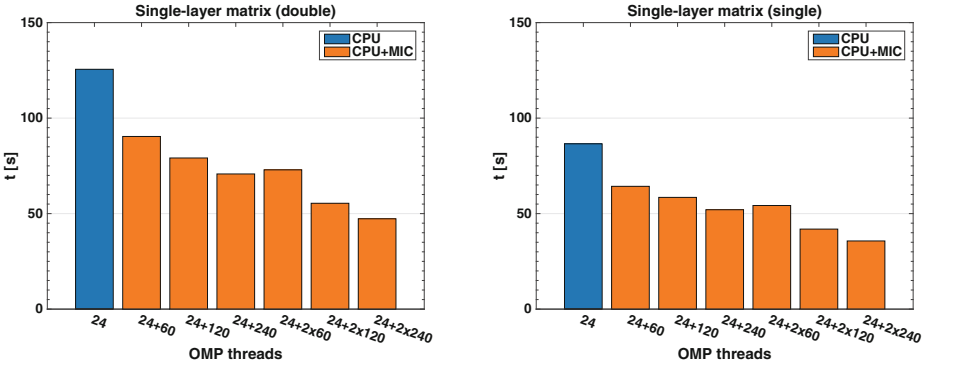

Fig. 2. Assembly of V_h accelerated by the Intel Xeon Phi coprocessors (Color figure online).

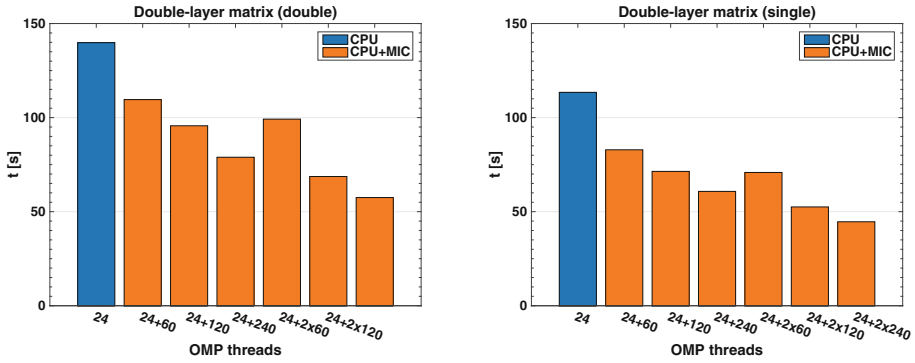

Fig. 3. Assembly of K_h accelerated by the Intel Xeon Phi coprocessors (Color figure online).

The coprocessor offers 1.2 TFLOPS of computing power, which is theoretically 1.26 times the power provided by the pair of the Intel Xeon processors (assuming 16 FMA instructions per cycle on both architectures [17]).

The experiments were performed using the Intel Compiler, version 15. The thread affinity on the coprocessor is set to `balanced`. We enable the offload runtime to allocate memory with 2 MB pages using the `MIC_USE_2MB_BUFFERS=100k` environmental variable. This significantly improves the performance since the coprocessors have to allocate data for large full system matrices. We test the assembly of the system matrices for a mesh with 81920 surface elements and 40962 nodes, thus $\mathsf{V}_h \in \mathbb{R}^{81920\times 81920}$ and $\mathsf{K}_h \in \mathbb{R}^{81920\times 40962}$.

The results of the numerical experiments are depicted in Figs. 2 and 3. We measure the assembly times on the coprocessor using the multiples of 60 threads in order to leave a core for the operating system. We compare the partially offloaded computation to the parallel assembly on 24 cores of the host's Intel Xeon processors. The workload was kept balanced among the coprocessors and CPU by adjusting the size of the matrix blocks.

Table 1. Speedup of CPU+MIC vs. CPU assembly for V_h.

# threads	$24+60$	$24+120$	$24+240$	$24+2\times 60$	$24+2\times 120$	$24+2\times 240$
Double	1.39	1.59	1.77	1.72	2.27	2.66
Single	1.35	1.48	1.66	1.60	2.07	2.43

Table 2. Speedup of CPU+MIC vs. CPU assembly for K_h.

# threads	$24+60$	$24+120$	$24+240$	$24+2\times 60$	$24+2\times 120$	$24+2\times 240$
Double	1.28	1.46	1.77	1.41	2.04	2.43
Single	1.37	1.59	1.87	1.60	2.16	2.54

Only using the physical cores of the coprocessor leads to a rather insignificant speedup of the offloaded computation. However, by utilizing the available logical cores of both cards the speedup improves significantly. The assembly time of the matrix V_h is reduced from 125.56 s (86.55 s) to 47.30 s (35.69 s) in the case of the double (single) precision arithmetic, respectively. Similarly, the computation time for the matrix K_h reduces from 139.83 s (113.43 s) to 57.50 s (44.64 s).

Tables 1 and 2 depict the speedups of the CPU+MIC vs. CPU computation (including the overhead caused by transfers between MIC and CPU). The necessity to employ more atomic operations in the assembly of K_h leads to only slightly worse results.

5 Conclusion

We have presented the boundary element method for the Laplace equation accelerated by the Intel Xeon Phi coprocessors. Sample codes describing the acceleration using the offload feature of the coprocessors were provided. The accelerated code featuring asynchronous CPU/coprocessor computation was tested on the Intel Xeon Phi 7120P cards in combination with two 12-core Intel Xeon processors. We have achieved a reasonable speedup in comparison with the non-accelerated code.

Although the numerical experiments were only performed for a relatively small problem, the full assembly of similar matrices can be used in boundary element tearing and interconnecting methods (BETI) [5,10], where the computational domain is divided into many small subdomains. The global algorithm usually requires accurate solution of local problems, which can be achieved by direct solvers with full matrices.

In addition to the classical BEM and BETI we turn our attention to the acceleration of the adaptive cross approximation method (ACA), which will enable the solution of engineering problems of large dimensions. Moreover, the presented code can be relatively easily extended to support the accelerated solution of the problems of elastostatics.

Acknowledgments. This work was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033). MM and JZ acknowledge the support of VŠB-TU Ostrava under the grant SGS SP2015/160. JJ was supported by the research project Architecture of parallel and embedded computer systems, Brno University of Technology, FIT-S-14-2297, 2014–2016 and by the SoMoPro II Programme co-financed by the European Union and the South-Moravian Region. This work reflects only the author's view and the European Union is not liable for any use that may be made of the information contained therein.

References

1. Bebendorf, M., Rjasanow, S.: Adaptive low-rank approximation of collocation matrices. Computing **70**(1), 1–24 (2003)
2. Dautray, R., Lions, J., Amson, J.: Mathematical Analysis and Numerical Methods for Science and Technology. Integral Equations and Numerical Methods, vol. 4. Springer, Heidelberg (1999)
3. Deslippe, J., Austin, B., Daley, C., Yang, W.S.: Lessons learned from optimizing science kernels for Intel's Knights Corner architecture. Comput. Sci. Eng. **17**(3), 30–42 (2015)
4. Dongarra, J., Gates, M., Haidar, A., Jia, Y., Kabir, K., Luszczek, P., Tomov, S.: HPC programming on intel many-integrated-core hardware with MAGMA port to Xeon Phi. Sci. Program. **2015**, 11 (2015)
5. Langer, U., Steinbach, O.: Boundary element tearing and interconnecting methods. Computing **71**(3), 205–228 (2003)
6. López-Portugués, M., López-Fernández, J., Díaz-Gracia, N., Ayestarán, R., Ranilla, J.: Aircraft noise scattering prediction using different accelerator architectures. J. Supercomputing **70**(2), 612–622 (2014). http://dx.doi.org/10.1007/s11227-014-1107-z
7. Lukáš, D., Kovář, P., Kovářová, T., Merta, M.: A parallel fast boundary element method using cyclic graph decompositions. Numer. Algorithms **70**, 807–824 (2015)
8. McLean, W.: Strongly Elliptic Systems and Boundary Integral Equations. Cambridge University Press, Cambridge (2000)
9. Merta, M., Zapletal, J.: Acceleration of boundary element method by explicit vectorization. Adv. Eng. Softw. **86**, 70–79 (2015)
10. Of, G., Steinbach, O.: The all-floating boundary element tearing and interconnecting method. J. Numer. Math. **17**(4), 277–298 (2009)
11. Of, G.: Fast multipole methods and applications. In: Schanz, M., Steinbach, O. (eds.) Boundary Element Analysis. Lecture Notes in Applied and Computational Mechanics, vol. 29, pp. 135–160. Springer, Heidelberg (2007)
12. Rjasanow, S., Steinbach, O.: The Fast Solution of Boundary Integral Equations. Springer, Heidelberg (2007)
13. Rokhlin, V.: Rapid solution of integral equations of classical potential theory. J. Comput. Phys. **60**(2), 187–207 (1985)
14. Sauter, S., Schwab, C.: Boundary Element Methods. Springer Series in Computational Mathematics. Springer, Heidelberg (2010)
15. Sirtori, S.: General stress analysis method by means of integral equations and boundary elements. Meccanica **14**(4), 210–218 (1979)
16. Steinbach, O.: Numerical Approximation Methods for Elliptic Boundary Value Problems: Finite and Boundary Elements. Texts in Applied Mathematics. Springer, Heidelberg (2008)
17. Intel Xeon Phi Coprocessor Peak Theoretical Maximums. http://www.intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-theoretical-maximums.html. Accessed 9 Oct 2015