BEM4I: A massively parallel boundary element solver

Michal Merta, Jan Zapletal, Michal Kravčenko, Lukáš Malý

michal.merta@vsb.cz, jan.zapletal@vsb.cz, michal.kravcenko@vsb.cz, lukas.maly@vsb.cz

IT4Innovations national01$#&0 supercomputing center@#01%101

BOUNDARY ELEMENT METHOD

The idea of the boundary element method (BEM) is to reformulate the volume PDE as an equivalent boundary integral equation.

$$Vt = \left(\frac{1}{2}I + K\right)u \quad \text{on } \partial\Omega$$

$$\Updownarrow$$

$$\begin{cases} -\Delta u - \kappa^2 u = 0 & \text{in } \Omega, \\ [u] = f & \text{on } \Gamma_{\mathrm{D}}, \\ [\partial u / \partial \boldsymbol{n}] = g & \text{on } \Gamma_{\mathrm{N}}. \end{cases}$$

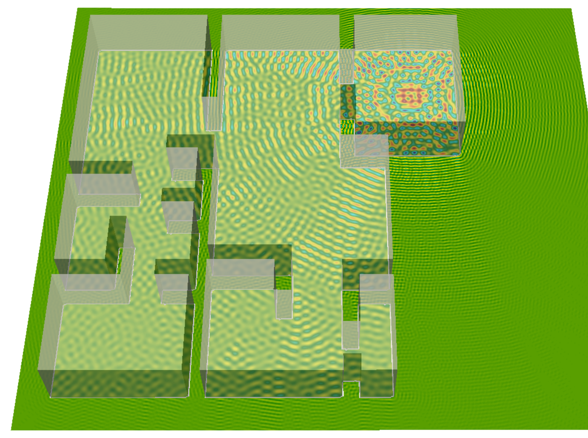


Fig. 1: Signal propagation in a building.

Reduction of a problem to the boundary of the domain makes the method well suited for problems on unbounded domains (e.g., **sound** or **electromagnetic wave scattering**) or **shape optimization problems**.

THE BEM4I LIBRARY

BEM4I is a library of parallel BEM solvers developed at IT4Innovations. It supports solution of the **Laplace**, **Lamé**, **Helmholtz**, and **wave** equations (see *Fig. 2*). To enable solution of large scale problems the **adaptive cross approximation** (ACA) is used.

To utilize modern HPC hardware we employ

- **OpenMP SIMD vectorization** for evaluation of singular integrals,
- **OpenMP threading** for local element contributions,
- **MPI-distributed** ACA,
- **MPI** for BETI (with the domain decomposition ESPRESO lib.),
- **offload** to Intel Xeon Phi coprocessors [1].

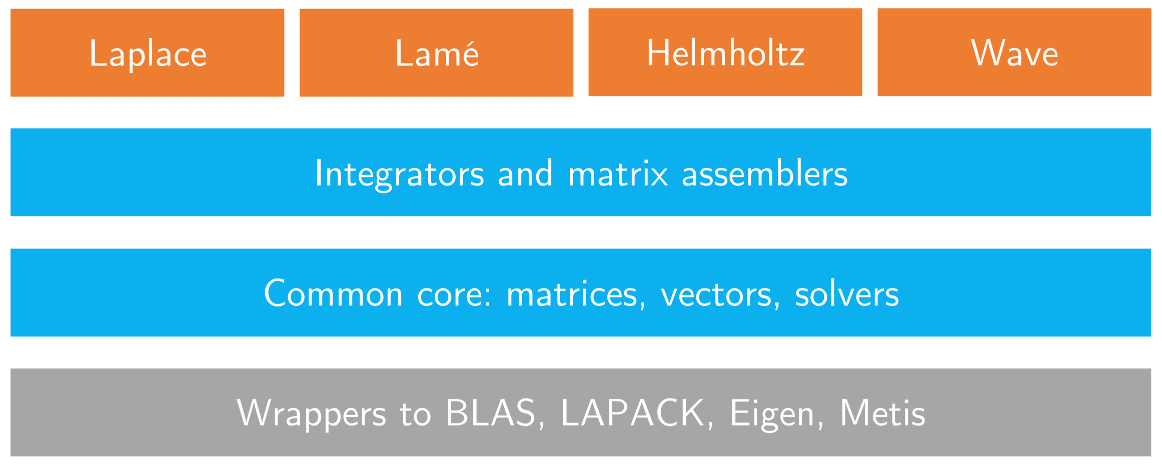
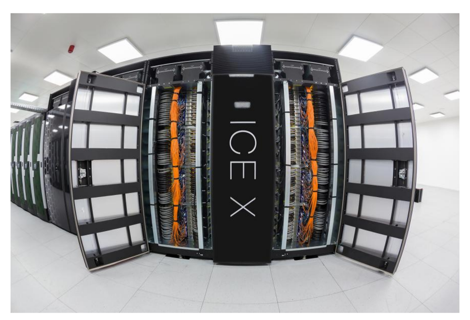


Fig. 2: Structure of the BEM4I library.

ENVIRONMENT

The numerical experiments were performed on the Salomon supercomputer at IT4Innovations, TDS at HLRN and Taurus at TU Dresden.



Salomon consists of 1008 compute nodes equipped with two Intel Xeon E5-2680v3 12-core processors and 128 GB of RAM. 432 nodes are accelerated by two Intel Xeon Phi 7120P. The nodes are interconnected by 7D enhanced hypercube InfiniBand network. The theoretical peak performance of the cluster is 2 Pflops.

TDS is a Cray CX machine consisting of 80 compute nodes equipped with Intel Xeon Phi 7250 Knights Landing 68-core processors, 16 GB of MCDRAM and 96 GB of DDR4 RAM. **Taurus** contains 32 nodes with 64-core Intel Xeon Phi 7210 and the same memory configuration.

SIMD VECTORIZATION OF NUMERICAL QUADRATURE

The implementation has to deal with matrices of the type

$$\mathsf{V}_h[\ell, k] := \frac{1}{4\pi} \int_{\tau_\ell} \int_{\tau_k} \frac{1}{\|\boldsymbol{x} - \boldsymbol{y}\|} \, \mathrm{d}s_{\boldsymbol{y}} \, \mathrm{d}s_{\boldsymbol{x}}.$$

BEM4I employs various techniques to efficiently utilize wide SIMD registers of modern CPUs [2,3]:

- OpenMP SIMD pragmas,
- data **alignment** and **padding**,
- AoS to **SoA** transition for spatial coordinates, complex numbers,
- **unit-strided** memory loads and stores.

AVOIDING EXPENSIVE MASKED OPERATIONS

In *Listing 1* we hint the strategy of avoiding costly masked calls of **sqrt** and **log** by masked evaluation of their arguments and dummy **tmp3=1.0**.

```

evaluatePrimitive( double s, ... ) {
...


 // do not add to f in special case
 if ( abs( s - sx ) > _EPS ) {
  if ( tmp2 < 0.0 ) {
   // masked evaluation of sqrt
   tmp3 = hh1 / ( sqrt(
    tmp1 * tmp1 + q_sq ) - tmp2 );
  } else {
   // masked evaluation of sqrt
   // same argument as above!
   tmp3 = tmp2 + sqrt(
    tmp1 * tmp1 + q_sq );
  }
  // masked evaluation of log
  f += ( s - sx ) * log( tmp3 );
}




...}
```

```
#pragma omp declare simd simdlen( 8 )
evaluatePrimitive( double s, ... ) {
...
 // unmasked evaluation of sqrt
 tmp1 = sqrt( tmp1 * tmp1 + q_sq );
 // do not add to f in special case
 if ( abs( s - sx ) > _EPS ) {
  if ( tmp2 < 0.0 ) {
   // masked division only
   tmp3 = hh1 / ( tmp1 - tmp2 );

  } else {

   // masked addition only
   tmp3 = tmp2 + tmp1;

  }


 } else {
  tmp3 = 1.0;
 }
 // unmasked evaluation of log
 f += ( s - sx ) * log( tmp3 );
...}
```

Listing 1. Scalar (left) and SIMD (right) evaluation of the primitive function.

Performance gain obtained for the assembly of two BEM matrices V_h, K_h and the evaluation of u_h is summarized in *Fig. 3* (right) and *Tab. 1*.

	scalar	AVX512(1)	AVX512(2)	AVX512(4)	AVX512(8)
V_h	1.00	1.39	1.29	2.91	7.68
K_h	1.00	1.62	1.72	3.41	8.25
u_h	1.00	1.52	1.54	3.71	11.84

Tab. 1. Speedup of OpenMP vectorized semi-analytic assembly vs. scalar version on Intel Xeon Phi 7210.

SHARED MEMORY PARALLELIZATION

Threading is employed at the level of local element contributions. In *Fig. 3* (left) see the scalability obtained on different architectures. Enforcing data locality and thread private buffers leads to optimal scaling up to tens or even hundreds of threads.

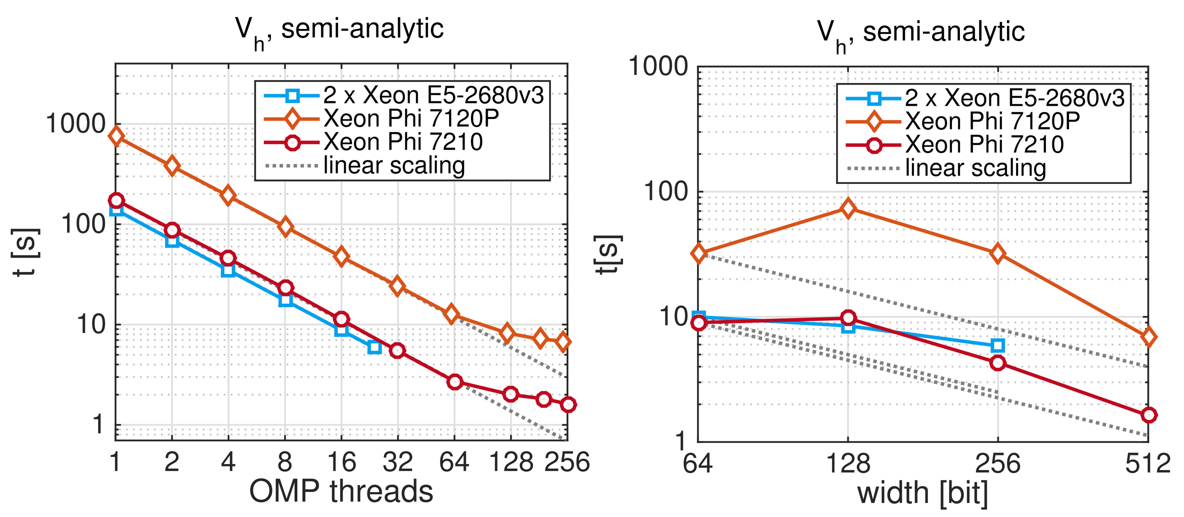
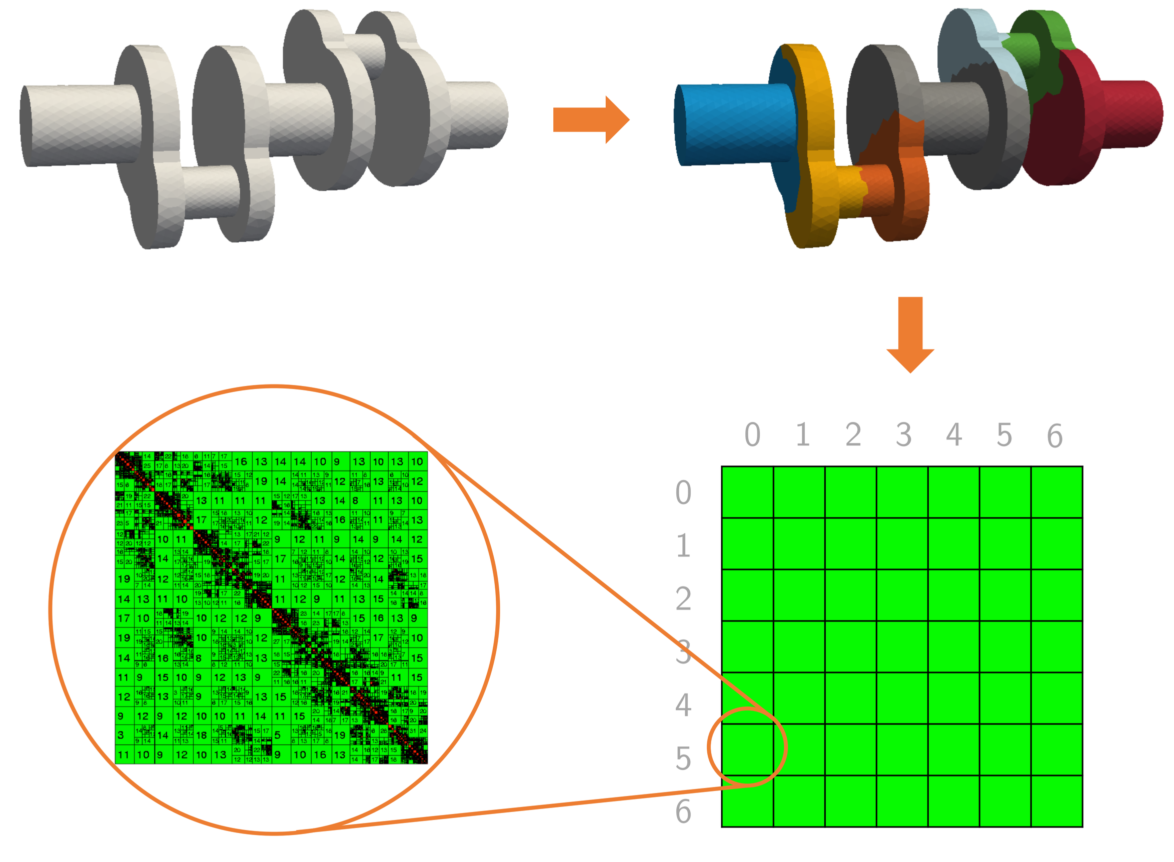


Fig. 3. Assembly times of OpenMP threaded and vectorized semi-analytic assembly vs. serial and scalar versions, respectively.

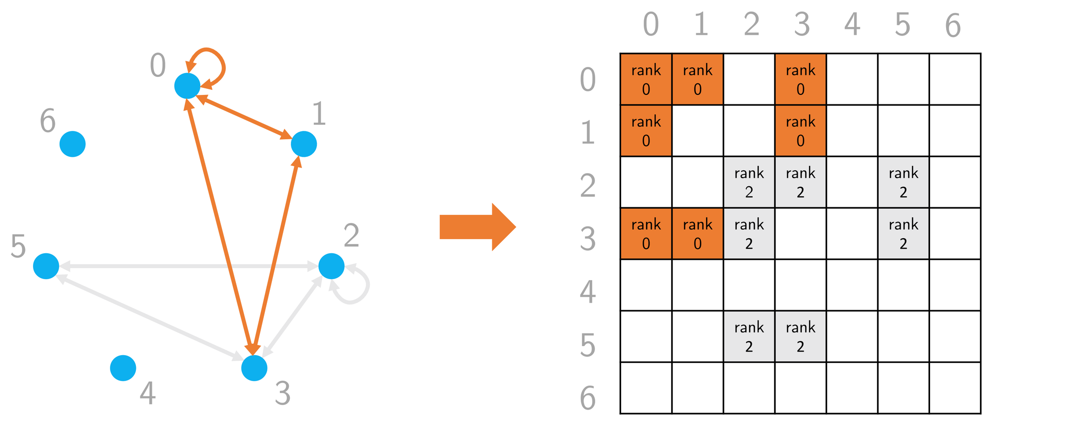
MASSIVELY PARALLEL ACA BEM

To sparsify full BEM matrices BEM4I implements a distributed version of ACA based on the approach from [4] distributing matrix blocks among nodes using **cyclic graph decompositions** (see *Fig. 4* and *5*). The method is well-suited for **homogeneous materials**.



Fig, 4. Decomposition of the original mesh induces block structure of the system matrix. Each block is approximated using low-rank matrix.

Matrix blocks are distributed among MPI ranks such that the number of submeshes per rank is minimized and each rank owns one diagonal block.

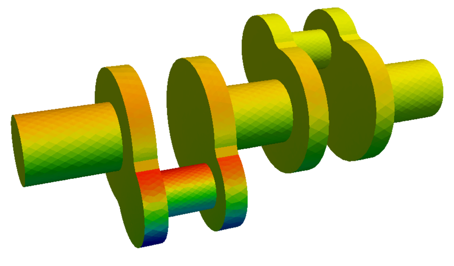


Fig, 5. Distribution of the blocks based on the cyclic graph decomposition.

In *Tab. 2* and *3* see results of experiments on the Salomon cluster.

	# nodes	8	16	32	64	128	256
V_h	assembly [s]	199.5	99.3	50.5	25.5	12.8	6.7
	efficiency [%]	100.0	100.4	98.5	97.7	97.4	93.0
K_h	assembly [s]	288.5	144.4	73.8	38.2	19.0	10.0
	efficiency [%]	100.0	99.9	97.7	94.4	94.9	90.1
CG	t/iter [s]	2.6	1.4	1.1	0.6	0.3	0.2
	efficiency [%]	100.0	92.9	59.0	54.1	54.2	40.6

Tab. 2. Strong scaling on a mesh with 1.5 million surface elements.



# surface elements	24.7 million	55.6 million
# nodes	64	256
# MPI ranks	128	512
assembly $\mathsf{V}_h/\mathsf{K}_h$ [s]	163.2/280.5	242.8/414.5
compr. $\mathsf{V}_h/\mathsf{K}_h$ [%]	0.081/0.163	0.092/0.185
RAM $\mathsf{V}_h/\mathsf{K}_h$ [GB]	3932/3987	22688/22873

Tab. 3. Distributed assembly of the system matrices V_h and K_h for the Laplace operator.

BOUNDARY ELEMENT TEARING AND INTERCONNECTING

The counterpart to the FETI DDM based on the BEM is the boundary element tearing and interconnecting (BETI) method.

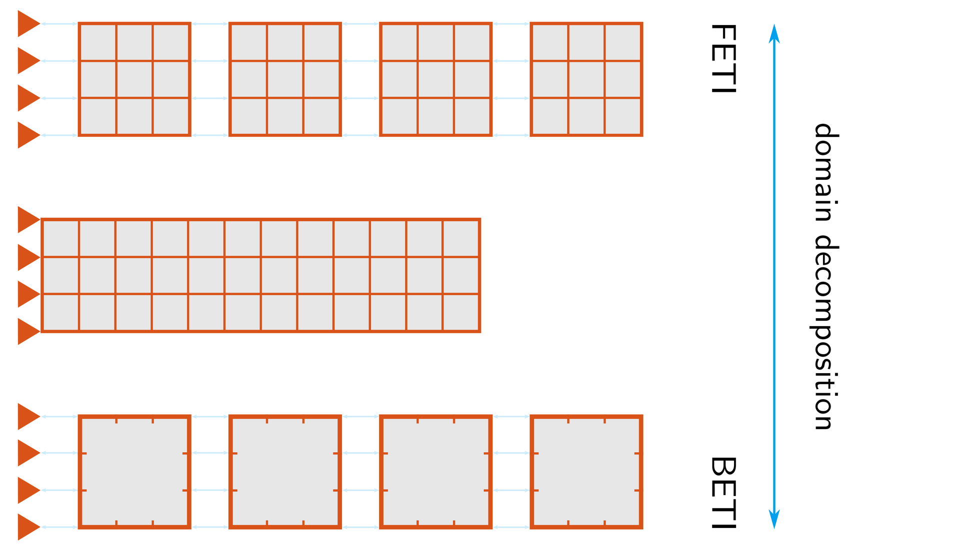
The local Dirichlet-to-Neumann maps are realized by the symmetric BEM-based Steklov-Poincaré operators

$$\mathsf{S}_h := \left(\frac{1}{2}\mathsf{M}_h + \mathsf{K}_h\right)^{\top} \mathsf{V}_h^{-1} \left(\frac{1}{2}\mathsf{M}_h + \mathsf{K}_h\right).$$

The method enables solution of problems with **non-homogeneous materials**.

BEM4I + ESPRESO = BETI

The ESPRESO library [5] provides an interface to the hybrid domain decomposition method (see *Fig. 6*). See *Fig. 7* and *8* for weak scalability experiments.



Fig, 6. FETI and BETI DDM methods decompose domain into smaller subdomains processed in parallel.

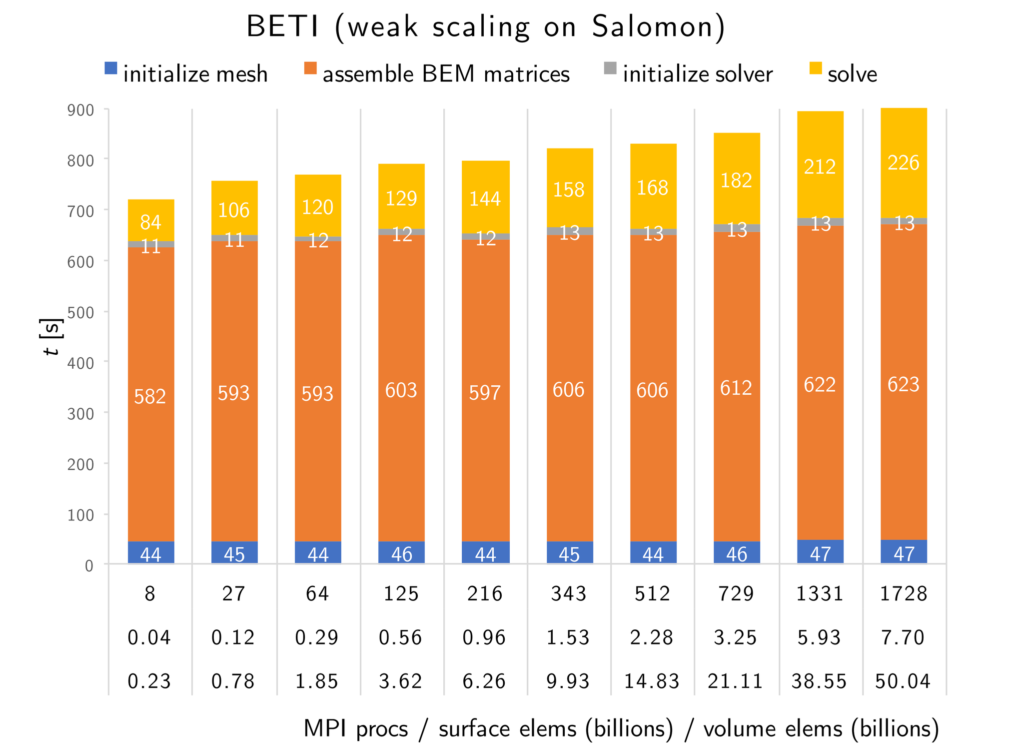


Fig. 7. Weak scaling of BETI on Salomon. The local problem is kept constant while scaling up to 1728 MPI processes on 864 compute nodes.

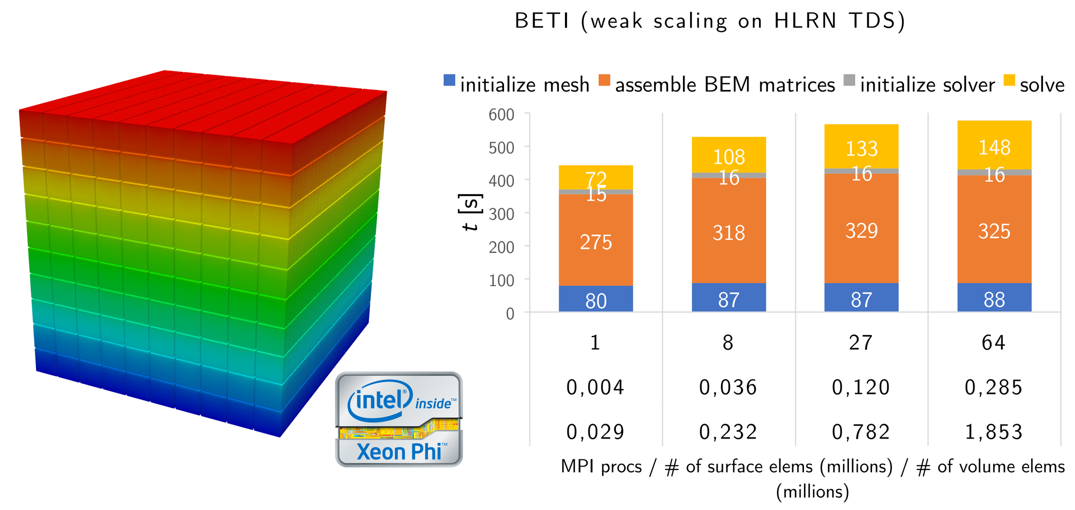


Fig. 8. Weak scaling of BETI on the HLRN TDS. The local problem is kept constant while scaling up to 64 MPI processes/nodes.

References

[1] Merta, M.; Zapletal, J.; Jaros, J. Many Core Acceleration of the Boundary Element Method. *LNCS 9611*, 2016.
[2] Zapletal, J.; Merta, M.; Malý, L. Boundary Element Quadrature Schemes for Multi- and Many-Core Architectures. *Comput. Math. Appl 74*, 2017.
[3] Zapletal, J.; Of, G., Merta, M. Parallel and vectorized implementation of analytic evaluation of boundary integral operators. Submitted.
[4] Lukas, D. et al. A parallel fast boundary element method using cyclic graph decompositions. Numer. Alg. 70, 2015.
[5] Riha, L. et al. Massively Parallel Hybrid Total FETI (HTFETI) Solver. PASC16, ACM, 2016.

bem4i.it4i.cz

Acknowledgments

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project 'IT4Innovations excellence in science – LQ1602' and from the Large Infrastructures for Research, Experimental Development and Innovations project 'IT4Innovations National Supercomputing Center – LM2015070' and by VSB-TU Ostrava under the grant SP2017/165.