Boundary Element Quadrature Schemes for Multi- and Many-Core Architectures

Jan Zapletal, Michal Merta, Lukáš Malý, Günther Of
jan.zapletal@bem4i.cz, michal.merta@bem4i.cz, lukas.maly@vsb.cz, guenther.of@univie.ac.at

Boundary Element method

The idea of the boundary element method (BEM) is to reformulate the volume PDE as an equivalent boundary integral equation, see Figure 1.

\[ \mathbf{V} = \left( \frac{1}{2} \mathbf{I} + \mathbf{K} \right) \mathbf{w} \text{ on } \Omega \]

- \[ \mathbf{K} \mathbf{w} = \mathbf{f} \text{ on } \Gamma _{D} \]
- \[ \mathbf{K} \mathbf{w} = \mathbf{g} \text{ on } \Gamma _{N} \]

To utilize modern HPC hardware we employ
- OpenMP SIMD vectorization for evaluation of simpler integrals,
- OpenMP threading for local element contributions,
- MPJ for BETI (with the domain decomposition ESPRESO lib.),
- offload to Intel Xeon coprocessors.

SIMD vectorization of semi-analytic evaluation

The semi-analytic assembly scheme leads to evaluation of

\[ \mathbf{V}_{\mathbf{a}}[\mathbf{h}] = \frac{1}{4\pi} \int _{\Omega} \left\{ \frac{1}{r} \text{ in } \Omega \right\} \text{d} s_{\mathbf{a},\mathbf{b}}. \]

BEM4I employs various techniques to efficiently utilize wide SIMD registers of modern CPUs:
- OpenMP-SIMD pragmas,
- data alignment and padding
- Alternative transition for spatial coordinates, complex numbers,
- unmasked memory loads and stores.

Avoiding expensive masked operations

In Listing 3 we want to avoid costly masked calls of sqrt and log by masking evaluation of their arguments and dummy temp=1.0.

Performance gain obtained for the assembly of two BEM matrices \( \mathbf{V}_{\mathbf{a}} \) and the evaluation of \( \mathbf{w} \) is summarized in Table 3 and Figure 2 (right).

SIMD vectorization of numerical evaluation

Second option is to use a series of transformations to render the integral results. This applies in 4D tensor Gauss quadrature

\[ \mathbf{V}_{\mathbf{b}}[\mathbf{h}] = \frac{1}{4\pi} \int _{\Omega} \left\{ \frac{1}{r} \text{ in } \Omega \right\} \text{d} s_{\mathbf{a},\mathbf{b}}. \]

Native implementation including four quadrature sums (see Listing 2) does not allow for efficient SIMD processing. We thus employ

- collapsing loops into a single one,
- preprocessing of data identical for all elements,
- data-decomposition to enable unmasked memory accesses.

Listing 2: Original scalar numerical assembly.

The optimizations lead to the code presented in Listing 3 and to the speedups summarized in Table 3 and Figure 3 (right). The absence of masked kernel evaluations leads to almost optimal scalability results.

Avoiding expensive masked operations

In Listing 3 we want to avoid costly masked calls of sqrt and log by masking evaluation of their arguments and dummy temp=1.0.

Performance gain obtained for the assembly of two BEM matrices \( \mathbf{V}_{\mathbf{a}} \) and the evaluation of \( \mathbf{w} \) is summarized in Table 3 and Figure 2 (right).

SIMD vectorization of numerical evaluation

Second option is to use a series of transformations to render the integral results. This applies in 4D tensor Gauss quadrature

\[ \mathbf{V}_{\mathbf{b}}[\mathbf{h}] = \frac{1}{4\pi} \int _{\Omega} \left\{ \frac{1}{r} \text{ in } \Omega \right\} \text{d} s_{\mathbf{a},\mathbf{b}}. \]

Native implementation including four quadrature sums (see Listing 2) does not allow for efficient SIMD processing. We thus employ

- collapsing loops into a single one,
- preprocessing of data identical for all elements,
- data-decomposition to enable unmasked memory accesses.

Listing 3: Vectorized numerical assembly.

Thresholding for numerical evaluation

OpenMP threading results in optimal speedups with respect to the serial version, see Table 3 and Figure 2 (left).

Thresholding for semi-analytic evaluation

Thresholding is employed at the level of local element contributions. In Table 2 and Figures 2 (left) the speedups the speedups obtained on different architectures. Enhancing data locality and thread private buffers leads to optimal scaling up to tens or even hundreds of threads.

Comparison of Xeon and Xeon Phi architectures

In Table 5 see the performance of BEM4I on the Knights Landing generation of Xeon Phi compared to the earlier Knights Corner coprocessor and multi-core dual-core Xeon CPU. Exploitation of the SIMD paradigm leads to almost optimal utilization of many-core CPUs.

Massively parallel BEM

The counterpart to the BETI domain decomposition method based on the boundary element method is the boundary element tearing and interconnecting (BETI) approach.

The local Dirichlet-to-Neumann maps are realized by the symmetric BEM-based Stencil-Functor operators.

BEM4I + ESPRESO = BETI

The ESPRESO library provides an interface to the hybrid domain decomposition method (see Figure 4). The connection between ESPRESO and BEM4I results in a massively parallel solver for large engineering problems. See Figures 3 and 6 for work scalability experiments.

![Figure 1](boundary.png)

**Figure 1:** Scalar and vectorized (right) evaluation of the primitive function. Masked evaluations of expensive functions are replaced by cheaper masked evaluations of their arguments.

![Figure 2](speedup.png)

**Figure 2:** Assembly times of OpenMP threaded and vectorized semi-analytic assembly vs. serial and vectoral versions, respectively.

![Figure 3](speedup.png)

**Figure 3:** Speedup of OpenMP vectorized numerical assembly vs. scalar version on Intel Xeon Phi 7250 (up to 8 double precision operands in 512-bit registers, 256 OpenMP threads).

![Figure 4](domain-decomposition.png)

**Figure 4:** Fronts and boundary element tearing and interconnecting methods (FETI, BETI) decompose domain into smaller subdomains processed in parallel and glue them together by Lagrange multipliers.

![Figure 5](scaling.png)

**Figure 5:** Weak scaling of BETI (left transfer) on Savonius equipped with dual-core Intel Xeon E5-2660v3 (Haswell). The local problem is kept constant while scaling up to 1920 MPI processes on 864 compute nodes.

![Figure 6](scaling.png)

**Figure 6:** Weak scaling of BETI (left transfer) on the ILDAP TDS equipped with Intel Xeon Phi 7250 (Knights Landing). The local problem is kept constant while scaling up to 64 MPI processes/nodes.

**References**


**Acknowledgments**

The 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 by the Large Infrastructures for Research, Experimental Development and Innovation project (IT4Innovations National Supercomputing Centre – LM2015070).