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

Jan Zapletal, Michal Merta, Lukáš Malý, Günther Of

jan.zapletal@vsb.cz, michal.merta@vsb.cz, lukas,maly@vsb.cz, of@tugraz.at

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, see *Figure 1.*

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

$$\Updownarrow$$

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

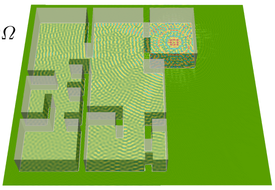

Figure 1. Solving the boundary integral equation is equivalent to solving the weak formulation of the considered scattering transmission problem.

BEM4I

BEM4I is a library of parallel BEM solvers developed at IT4Innovations. The implementation has to deal with matrices of the type

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

To utilize modern HPC hardware we employ

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

SIMD vectorization of semi-analytic evaluation

The semi-analytic assembly scheme leads to evaluation of

$$\mathsf{V}_h[\ell, k] \approx \Delta_\ell \sum_m w_m \frac{1}{4\pi}\int_{\tau_k}\frac{1}{\|\boldsymbol{x}^m-\boldsymbol{y}\|}\,\mathrm{d}s_{\boldsymbol{y}}.$$

BEM4I employs various techniques to efficiently utilize wide SIMD registers of modern CPUs:

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

```
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 vectorized (right) evaluation of the primitive function. Masked evaluations of expensive functions are replaced by cheaper masked evaluations of their arguments.

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`.

Performance gain obtained for the assembly of two BEM matrices V_h, K_h and the evaluation of u_h is summarized in *Table 1* and *Figure 2* (right).

t [s]	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

Table 1. Speedup of OpenMP vectorized semi-analytic assembly vs. scalar version on Intel Xeon Phi 7210 (up to 8 double precision operands in 512-bit registers, 256 OpenMP threads).

Threading for semi-analytic evaluation

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

t [s]	serial	64 th.	128 th.	192 th.	256 th.
V_h	1.00	64.57	87.82	96.50	108.41
K_h	1.00	62.56	84.11	87.06	94.59
u_h	1.00	63.42	81.89	78.44	86.15

Table 2. Speedup of OpenMP threaded semi-analytic assembly vs. serial version on Intel Xeon Phi 7210 (up to 4-way hyper-threading, OpenMP SIMD with AVX512).

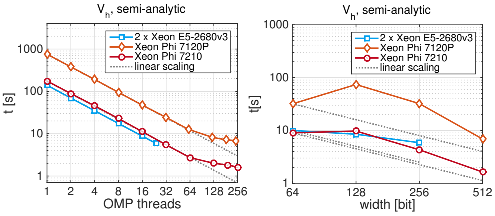

Figure 2. Assembly times of OpenMP threaded and vectorized semi-analytic assembly vs. serial and scalar versions, respectively.

SIMD vectorization of numerical evaluation

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

$$\mathsf{V}_h[\ell, k] \approx \sum_s\sum_m\sum_n\sum_o\sum_p w_m w_n w_o w_p$$
$$\hat{k}(\boldsymbol{F}^s(\eta_{1,m}, \eta_{2,n}, \eta_{3,o}, \xi_p))\boldsymbol{S}^s(\eta_{1,m}, \eta_{2,n}, \eta_{3,o}, \xi_p).$$

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

- **collapsing** of the loops into a single one,
- **precomputation** of data identical for all elements,
- data **duplication** to ensure **unit-strided** memory accesses.

```
for( int m = 0; m < S1; ++m ){
 for( int n = 0; n < S2; ++n ){
  for( int o = 0; o < S3; ++o ){
   for( int p = 0; p < S4; ++p ){
    for( int s = 0; s < S; ++s ){
     // map from hypercube to (tau x tau), get nu, mu, jac
     cubeToRef( eta1[ m ], eta2[ n ], eta3[ o ], ksi[ p ],
      s, nu, mu, jac );
     // map from (tau x tau) to (tau_l x tau_k), get x, y
     refToTri( xl1, ..., yk3, nu, mu, x, y );
     // multiply kernel with weights and jacobian
     kernel = jac * w1[ m ] * w2[ n ] * w3[ o ] * w4[ p ]
      * evalSingleLayerKernel( x, y );
     entry += kernel;
} } } } }
```

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.

```
for( int s = 0; s < S; ++s ){
 // map from (tau x tau) to (tau_l x tau_k), get x, y in SoA format
 refToTri( s, xl1, ..., yk3, nu1, ..., mu2, x1, ..., y3 );
#pragma omp simd \\
aligned( jacWV, x1, ..., y3 ) \\
reduction( + : entry ) \\
simdlen( 8 )
 for ( int c = 0; c < S1*S2*S3*S4; ++c ) { // collapsed loop
  // multiply kernel with weights and jacobian, unit-strided access
  kernel = jacWV[ c ] *
   evalSingleLayerKernel( x1[ c ], x2[ c ], x3[ c ],
                          y1[ c ], y2[ c ], y3[ c ] );
  entry += kernel;
} }
```

Listing 3. Vectorized numerical assembly.

t [s]	scalar	AVX512(1)	AVX512(2)	AVX512(4)	AVX512(8)
V_h	1.00	2.00	3.78	6.07	7.62
K_h	1.00	1.20	2.26	3.89	5.53

Table 3. Speedup of OpenMP vectorized numerical assembly vs. scalar version on Intel Xeon Phi 7210 (up to 8 double precision operands in 512-bit registers, 256 OpenMP threads).

Threading for numerical evaluation

OpenMP threading results in optimal speedups with respect to the serial version, see *Table 4* and *Figure 3* (left).

t [s]	serial	64 th.	128 th.	192 th.	256 th.
V_h	1.00	62.26	62.89	54.22	57.65
K_h	1.00	62.76	73.64	67.76	73.64

Table 4. Speedup of OpenMP threaded numerical assembly vs. serial version on Intel Xeon Phi 7210 (up to 4-way hyper-threading, OpenMP SIMD with AVX512).

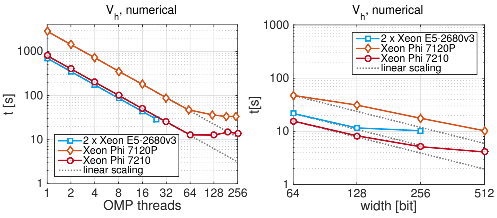

Figure 3. Assembly times of OpenMP threaded and vectorized numerical assembly vs. serial and scalar versions, respectively.

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-socket Haswell CPU. Exploitation of the SIMD paradigm leads to almost optimal utilization of many-core CPUs.

	Xeon E5-2680v3		Xeon Phi 7120P	
	semi-analytic	numerical	semi-analytic	numerical
V_h	3.62	2.51	4.23	2.51
K_h	3.37	1.86	4.24	2.76
u_h	4.25	---	5.01	---

Table 5. Speedup of the semi-analytic and numerical assembly on Intel Xeon Phi 7210 vs. dual-socket Xeon E5-2680v3 and Xeon Phi 7120P.

Massively parallel BEM

The counterpart to the FETI 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 Steklov-Poincaré operators

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

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 5* and *6* for weak scalability experiments.

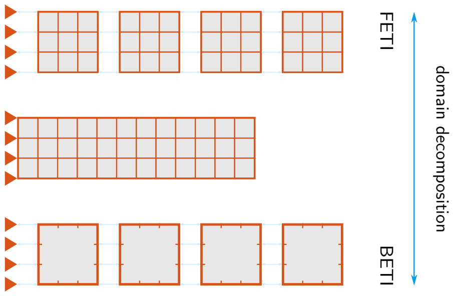

Figure 4. Finite and boundary element tearing and interconnecting methods (FETI, BETI) decompose domain into smaller subdomains processed in parallel and glue them together by Lagrange multipliers.

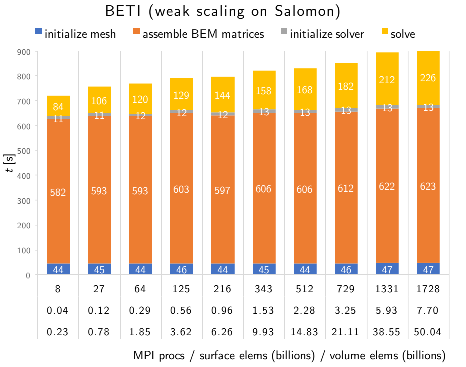

Figure 5. Weak scaling of BETI (heat transfer) on Salomon equipped with dual-socket Intel Xeon E5-2680v3 (Haswell). The local problem is kept constant while scaling up to 1728 MPI processes on 864 compute nodes.

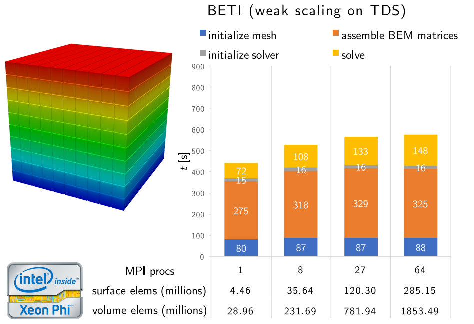

Figure 6. Weak scaling of BETI (heat transfer) on the HLRN TDS equipped with Intel Xeon Phi 7250 (Knights Landing). The local problem is kept constant while scaling up to 64 MPI processes/nodes.

References

[1] Merta, M.; Zapletal, J.; Jaros, J. Many Core Acceleration of the Boundary Element Method. *LNCS*, 2016, 116-125.
[2] Zapletal, J.; Merta, M.; Malý, L. Boundary Element Quadrature Schemes for Multi- and Many-Core Architectures. *Comput. Math. Appl*, 2017, 74, 157-173.
[3] Zapletal, J.; Of, G., Merta, M. Parallel and vectorized implementation of analytic evaluation of boundary integral operators. Work in progress.
[4] Riha, L. et al. Massively Parallel Hybrid Total FETI (HTFETI) Solver. ACM, 2016.

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’.

IT4Innovations, VŠB – Technical University of Ostrava, Czech Republic

http://bem4i.it4i.cz/