Boundary element quadrature schemes for multi- and many-core architectures

Jan Zapletal, Michal Merta, Lukáš Malý

IT4Innovations, Dept. of Applied Mathematics
VŠB-TU Ostrava

jan.zapletal@vsb.cz

Intel MIC programming workshop, February 7, 2017

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

EUROPEAN UNION
EUROPEAN REGIONAL DEVELOPMENT FUND
INVESTING IN YOUR FUTURE

2007–13
OP Research and Development for Innovation

1. BEM4I
   - Boundary element method
   - OpenMP threading
   - OpenMP vectorization
   - Adaptive cross approximation
2. Numerical experiments
   - Full assembly
   - ACA assembly
3. Conclusion

BEM4I

- BEM is an alternative to FEM for the solution of PDEs,
- BEM4I - 3D boundary element library for
  - Laplace equation (heat transfer),
  - Helmholtz equation (wave scattering),
  - Lamé equation (linear elasticity),
  - time-dependent wave equation.
- Specifications
  - C++, interface to MKL or other BLAS, LAPACK,
  - ACA for matrix sparsification.
- Strategies
  - SIMD vectorization for surface integrals (**OpenMP**, Vc library),
  - OpenMP for local element contributions / individual ACA blocks,
  - MPI for distributed matrices,
  - BETI with the ESPRESO library,
  - Intel MIC offload/**native**.

BEM4I

- BEM is an alternative to FEM for the solution of PDEs,
- BEM4I - 3D boundary element library for
  - Laplace equation (heat transfer),
  - Helmholtz equation (wave scattering),
  - Lamé equation (linear elasticity),
  - time-dependent wave equation.
- Specifications
  - C++, interface to MKL or other BLAS, LAPACK,
  - ACA for matrix sparsification.
- Strategies
  - SIMD vectorization for surface integrals (**OpenMP**, Vc library),
  - OpenMP for local element contributions / individual ACA blocks,
  - MPI for distributed matrices,
  - BETI with the ESPRESO library,
  - Intel MIC offload/**native**.

BEM4I

- BEM is an alternative to FEM for the solution of PDEs,
- BEM4I - 3D boundary element library for
  - Laplace equation (heat transfer),
  - Helmholtz equation (wave scattering),
  - Lamé equation (linear elasticity),
  - time-dependent wave equation.
- Specifications
  - C++, interface to MKL or other BLAS, LAPACK,
  - ACA for matrix sparsification.
- Strategies
  - SIMD vectorization for surface integrals (**OpenMP**, Vc library),
  - OpenMP for local element contributions / individual ACA blocks,
  - MPI for distributed matrices,
  - BETI with the ESPRESO library,
  - Intel MIC offload/**native**.

BEM4I

- BEM is an alternative to FEM for the solution of PDEs,
- BEM4I - 3D boundary element library for
  - Laplace equation (heat transfer),
  - Helmholtz equation (wave scattering),
  - Lamé equation (linear elasticity),
  - time-dependent wave equation.
- Specifications
  - C++, interface to MKL or other BLAS, LAPACK,
  - ACA for matrix sparsification.
- Strategies
  - SIMD vectorization for surface integrals (**OpenMP**, Vc library),
  - OpenMP for local element contributions / individual ACA blocks,
  - MPI for distributed matrices,
  - BETI with the ESPRESO library,
  - Intel MIC offload/**native**.

Boundary value problem for the Laplace equation

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

Representation formula for $\boldsymbol{x} \in \Omega$

$$u(\boldsymbol{x}) = \frac{1}{4\pi} \int_{\partial\Omega} \frac{1}{\|\boldsymbol{x}-\boldsymbol{y}\|} \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} - \frac{1}{4\pi} \int_{\partial\Omega} \frac{\langle \boldsymbol{x}-\boldsymbol{y}, \boldsymbol{n}(\boldsymbol{y}) \rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|^3} u(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}.$$

Boundary integral equations for $\boldsymbol{x} \in \partial\Omega$

$$\frac{1}{4\pi} \int_{\partial\Omega} \frac{1}{\|\boldsymbol{x}-\boldsymbol{y}\|} \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} = \frac{1}{2} u(\boldsymbol{x}) + \frac{1}{4\pi} \int_{\partial\Omega} \frac{\langle \boldsymbol{x}-\boldsymbol{y}, \boldsymbol{n}(\boldsymbol{y}) \rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|^3} u(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}},$$

$$-\frac{\partial}{\partial \boldsymbol{n}_{\boldsymbol{x}}} \frac{1}{4\pi} \int_{\partial\Omega} \frac{\langle \boldsymbol{x}-\boldsymbol{y}, \boldsymbol{n}(\boldsymbol{y}) \rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|^3} u(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}} = \frac{1}{2} \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{x}) - \frac{1}{4\pi} \int_{\partial\Omega} \frac{\langle \boldsymbol{y}-\boldsymbol{x}, \boldsymbol{n}(\boldsymbol{x}) \rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|^3} \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}.$$

Discretization leads to the systems

$$\mathsf{V}_h \boldsymbol{g} = \left(\frac{1}{2}\mathsf{M}_h + \mathsf{K}_h\right)\boldsymbol{f}, \qquad \mathsf{D}_h \boldsymbol{f} = \left(\frac{1}{2}\mathsf{M}_h - \mathsf{K}_h\right)^{\top}\boldsymbol{g}$$

with the matrices

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

$$\mathsf{K}_h[\ell,i] := \frac{1}{4\pi}\int_{\tau_\ell}\int_{\partial\Omega}\varphi_i(\boldsymbol{y})\frac{\langle\boldsymbol{x}-\boldsymbol{y},\boldsymbol{n}(\boldsymbol{y})\rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|^3}\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{x}}$$

$$\mathsf{D}_h[j,i] := \frac{1}{4\pi}\int_{\partial\Omega}\int_{\partial\Omega}\frac{\langle\mathbf{curl}\,\varphi_i(\boldsymbol{y}),\mathbf{curl}\,\varphi_j(\boldsymbol{x})\rangle}{\|\boldsymbol{x}-\boldsymbol{y}\|}\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{x}} = \mathsf{T}_h^{\top}\,\mathrm{diag}(\mathsf{V}_h,\mathsf{V}_h,\mathsf{V}_h)\mathsf{T}_h,$$

$$\mathsf{M}_h[\ell,i] := \int_{\tau_\ell}\varphi_i(\boldsymbol{x})\,\mathrm{d}\boldsymbol{s}_{\boldsymbol{x}}.$$

- OpenMP threading for V_h

```
#pragma omp parallel for
for( int tau_k = 0; tau_k < E; ++tau_k ){ // columns
 for( int tau_l = 0; tau_l < E; ++tau_l ){ // rows
  SLIntegrator.getLocalMatrix( *tau_l, *tau_k, Vloc );
  V.set( *tau_l, *tau_k, Vloc.get( 0, 0 ) );
 } }
```

- OpenMP threading for K_h

- OpenMP threading for V_h

```
#pragma omp parallel for
for( int tau_k = 0; tau_k < E; ++tau_k ){ // columns
 for( int tau_l = 0; tau_l < E; ++tau_l ){ // rows
  SLIntegrator.getLocalMatrix( *tau_l, *tau_k, Vloc );
  V.set( *tau_l, *tau_k, Vloc.get( 0, 0 ) );
 } }
```

- OpenMP threading for K_h

```
#pragma omp parallel for
for( int tau_k = 0; tau_k < E; ++tau_k ){ // columns
 for( int tau_l = 0; tau_l < E; ++tau_l ){ // rows
  DLIntegrator.getLocalMatrix( *tau_l, *tau_k, Kloc );
#pragma omp atomic // (inside of add_atomic)
  K.add_atomic( *tau_l, tau_k->node[ 0 ], Kloc.get( 0, 0 ) );
#pragma omp atomic // (inside of add_atomic)
  K.add_atomic( *tau_l, tau_k->node[ 1 ], Kloc.get( 0, 1 ) );
#pragma omp atomic // (inside of add_atomic)
  K.add_atomic( *tau_l, tau_k->node[ 2 ], Kloc.get( 0, 2 ) );
 } }
```

- OpenMP threading for V_h

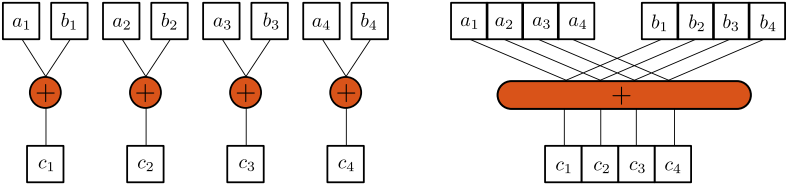
```
#pragma omp parallel for
for( int tau_k = 0; tau_k < E; ++tau_k ){ // columns
 for( int tau_l = 0; tau_l < E; ++tau_l ){ // rows
  SLIntegrator.getLocalMatrix( *tau_l, *tau_k, Vloc );
  V.set( *tau_l, *tau_k, Vloc.get( 0, 0 ) );
 } }
```

- OpenMP threading for K_h

```
#pragma omp parallel for
for( int tau_l = 0; tau_l < E; ++tau_l ){ // rows
 for( int tau_k = 0; tau_k < E; ++tau_k ){ // columns
  DLIntegrator.getLocalMatrix( *tau_l, *tau_k, Kloc );

  K.add( *tau_l, tau_k->node[ 0 ], Kloc.get( 0, 0 ) );

  K.add( *tau_l, tau_k->node[ 1 ], Kloc.get( 0, 1 ) );

  K.add( *tau_l, tau_k->node[ 2 ], Kloc.get( 0, 2 ) );
 } }
```

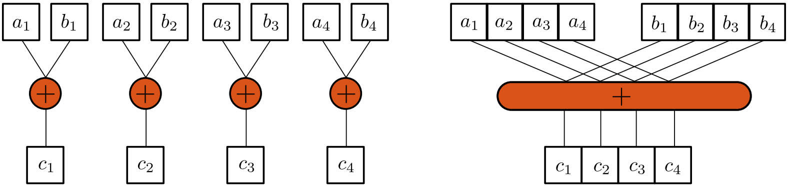
- Avoid `#pragma omp critical`, use `#pragma omp atomic` if applicable.

- Single Instruction Multiple Data (SIMD)
  - processing vector with a single operation,
  - provides data level parallelism,
  - elements are of the same type.



- Vector length
  - 256 bits for AVX-2 (Haswell), 4 DP operands,
  - 512 bits for IMCI (KNC), AVX512 (KNL), 8 DP operands.
- Vectorization achieved by
  - compiler auto-vectorization (code refactoring can help),
  - **OpenMP 4.0 pragmas** (`#pragma omp simd`),
  - intrinsic functions,
  - wrapper library (Vc),
  - assembly.

- Single Instruction Multiple Data (SIMD)
  - processing vector with a single operation,
  - provides data level parallelism,
  - elements are of the same type.



- Vector length
  - 256 bits for AVX-2 (Haswell), 4 DP operands,
  - 512 bits for IMCI (KNC), AVX512 (KNL), 8 DP operands.
- Vectorization achieved by
  - compiler auto-vectorization (code refactoring can help),
  - **OpenMP 4.0 pragmas** (`#pragma omp simd`),
  - intrinsic functions,
  - wrapper library (Vc),
  - assembly.

- Tuned vectorized integration over a square

```
double * w = new double[ S ];
w[ 0 ] = ... // init. weights
// x^1_1, x^1_2, ..., x^S_1, x^S_2
double x [ ] = { ... };






for( int l = 0; l < S; ++l ){
  result += w[ l ] * f( x[ 2 * l ], x[ 2 * l + 1 ] );
}

delete w;
```

- Loop peeling & strided access to memory

- Tuned vectorized integration over a square

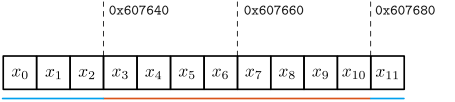
```
double * w = new double[ S ];
w[ 0 ] = ... // init. weights
// x^1_1, x^1_2, ..., x^S_1, x^S_2
double x [ ] = { ... };





#pragma omp simd reduction( + : result )
for( int l = 0; l < S; ++l ){
  result += w[ l ] * f( x[ 2 * l ], x[ 2 * l + 1 ] );
}

delete w;
```

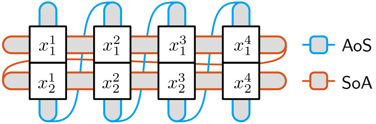
- Loop peeling & strided access to memory

Tuned vectorized integration over a square

```
double * w = new double[ S ];
w[ 0 ] = ... // init. weights
// x^1_1, x^1_2, ..., x^S_1, x^S_2
double x [ ] = { ... };

#pragma omp simd reduction( + : result )
for( int l = 0; l < S; ++l ){
  result += w[ l ] * f( x[ 2 * l ], x[ 2 * l + 1 ] );
}

delete w;
```

Loop peeling & strided access to memory

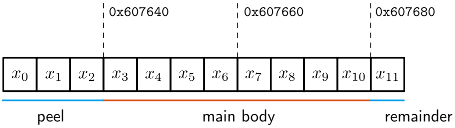


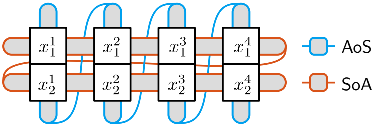


Tuned vectorized integration over a square

```
double * w = (double *) _mm_malloc( S * sizeof(double), 64 );
w[ 0 ] = ... // init. weights
// x^1_1, ..., x^S_1
double x1 [ ]  __attribute__( ( aligned( 64 ) ) )  = { ... };
// x^1_2, ..., x^S_2
double x2 [ ]  __attribute__( ( aligned( 64 ) ) )  = { ... };

__assume_aligned( w, 64 ); // tell compiler about alignment

#pragma omp simd reduction( + : result )
for( int l = 0; l < S; ++l ){
  result += w[ l ] * f( x1[ l ], x2[ l ] );
}

_mm_free( w );
```

Loop peeling & strided access to memory



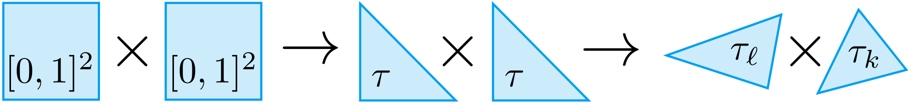


Duffy substitution for $\tau_\ell \times \tau_k$,

$$\mathsf{V}_h[\ell, k] = \sum_s \int_0^1 \int_0^1 \int_0^1 \int_0^1 k(\boldsymbol{F}^s(\eta_1, \eta_2, \eta_3, \xi))\boldsymbol{S}^s(\eta_1, \eta_2, \eta_3, \xi)\,\mathrm{d}\eta_1\,\mathrm{d}\eta_2\,\mathrm{d}\eta_3\,\mathrm{d}\xi$$

with $\boldsymbol{F}^s\colon [0,1]^4 \to S \subset \tau_\ell \times \tau_k$,

$$\boldsymbol{F}^s(\eta_1, \eta_2, \eta_3, \xi) = (\boldsymbol{x}, \boldsymbol{y}), \qquad \boldsymbol{S}^s(\eta_1, \eta_2, \eta_3, \xi)\,\mathrm{d}\eta_1\,\mathrm{d}\eta_2\,\mathrm{d}\eta_3\,\mathrm{d}\xi = \mathrm{d}\boldsymbol{s}_{\boldsymbol{x}}\mathrm{d}\boldsymbol{s}_{\boldsymbol{y}}.$$



Approximated by tensor Gauss quadrature

$$\mathsf{V}_h[\ell, k] \approx \sum_s \sum_m w_m \sum_n w_n \sum_o w_o \sum_p w_p\, k(\boldsymbol{F}^s(x_m, x_n, x_o, x_p))\boldsymbol{S}^s(x_m, x_n, x_o, x_p).$$

- Collapsed integration loop in getLocalMatrix.

```
__assume_aligned( x1ss, 64 ); // all data aligned
...

switch( type ){
 case( identicalElements ):
  for( int simplex = 0; simplex < 6; ++simplex ){

  refToTri( simplex, x1, ..., y3, x1ref, ..., y2ref, x1ss,
    ..., y3ss );

#pragma omp simd reduction( + : entry )
   for ( c = 0; c < S1*S2*S3*S4; ++c ) { // collapsed
    kernel = weights_jacV[ c ]
      * evalSingleLayerKernel( x1ss[ c ], x2ss[ c ],
        x3ss[ c ],  y1ss[ c ], y2ss[ c ], y3ss[ c ] );
    entry += kernel;
   } }
  break;
 ... // quadrature over other pairs of elements
}
```

- SIMD evaluation of quadrature points in `refToTri`.

```
__assume_aligned( x1ss, 64 ); // all data aligned
...

#pragma omp simd
for( int c = 0; c < S1*S2*S3*S4; ++c ){
 x1ss[ c ] = x1[ 0 ]
  + ( x2[ 0 ] - x1[ 0 ] ) * x1ref[ simplex ][ c ]
  + ( x3[ 0 ] - x1[ 0 ] ) * x2ref[ simplex ][ c ];
  ... // compute x2ss, x3ss, y1ss, y2ss, y3ss
}
```

- SIMD evaluation of the kernel in `evalSingleLayerKernel`.

```
#pragma omp declare simd
double evalSingleLayerKernel(
 double x1, double x2, double x3,
 double y1, double y2, double y3
) const {

 double d1 = x1 - y1, d2 = x2 - y2, d3 = x3 - y3;
 double norm = sqrt( d1 * d1 + d2 * d2 + d3 * d3 );

 return ( 1 / ( norm * 4.0 * M_PI ) );
}
```

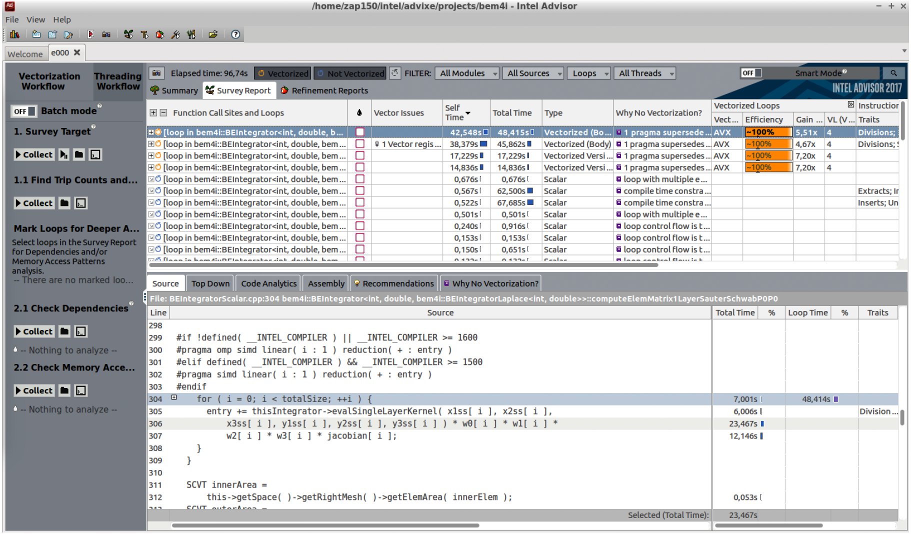
- SIMD evaluation of quadrature points in `refToTri`.

```
__assume_aligned( x1ss, 64 ); // all data aligned
...

#pragma omp simd
for( int c = 0; c < S1*S2*S3*S4; ++c ){
 x1ss[ c ] = x1[ 0 ]
  + ( x2[ 0 ] - x1[ 0 ] ) * x1ref[ simplex ][ c ]
  + ( x3[ 0 ] - x1[ 0 ] ) * x2ref[ simplex ][ c ];
  ... // compute x2ss, x3ss, y1ss, y2ss, y3ss
}
```

- SIMD evaluation of the kernel in `evalSingleLayerKernel`.

```
#pragma omp declare simd
double evalSingleLayerKernel(
 double x1, double x2, double x3,
 double y1, double y2, double y3
) const {

 double d1 = x1 - y1, d2 = x2 - y2, d3 = x3 - y3;
 double norm = sqrt( d1 * d1 + d2 * d2 + d3 * d3 );

 return ( 1 / ( norm * 4.0 * M_PI ) );
}
```

Intel Advisor



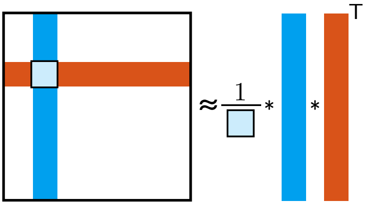
Intel compiler reports

- -qopt-report=5 -qopt-report-phase=vec

```
LOOP BEGIN at BEIntegratorScalar.cpp(304,5)
   remark 15340: pragma supersedes previous setting [
    BEIntegratorScalar.cpp(300,1) ]
   remark 15388: vectorization support: reference x1ss[i] has
    aligned access   [ BEIntegratorScalar.cpp(305,55) ]
   ... // all data reported as aligned
   remark 15305: vectorization support: vector length 4
   remark 15309: vectorization support: normalized
    vectorization overhead 0.443
   remark 15301: OpenMP SIMD LOOP WAS VECTORIZED
   remark 15448: unmasked aligned unit stride loads: 16
   remark 15475: --- begin vector cost summary ---
   remark 15476: scalar cost: 97
   remark 15477: vector cost: 17.500
   remark 15478: estimated potential speedup: 5.510
   remark 15488: --- end vector cost summary ---
LOOP END
```

Adaptive cross approximation

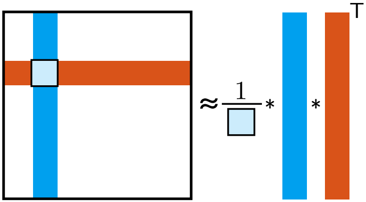
- Complexity $\mathcal{O}(n^2) \to \mathcal{O}(n \log n)$,
- mesh divided into clusters,
- *non-admissible* clusters assembled in full,
- *admissible* clusters approximated as $\mathsf{C} \approx \mathsf{U}\mathsf{V}^\top$,
- assembly of clusters distributed by OpenMP.



```
#pragma omp parallel for
for( int i = 0; i < n_nonadmissible_blocks; ++i ){
  getNonadmissibleBlock( i, localBlock );
  ACAMatrix.addNonadmissibleBlock( i, localBlock );
}

#pragma omp parallel for
for( int i = 0; i < n_admissible_blocks; ++i ){
  getAdmissibleBlock( i, localBlock );
  ACAMatrix.addAdmissibleBlock( i, localBlock );
}
```

Adaptive cross approximation

- Complexity $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n \log n)$,
- mesh divided into clusters,
- *non-admissible* clusters assembled in full,
- *admissible* clusters approximated as $\mathsf{C} \approx \mathsf{U}\mathsf{V}^\top$,
- assembly of clusters distributed by OpenMP.




```
#pragma omp parallel for
for( int i = 0; i < n_nonadmissible_blocks; ++i ){
  getNonadmissibleBlock( i, localBlock );
  ACAMatrix.addNonadmissibleBlock( i, localBlock );
}

#pragma omp parallel for
for( int i = 0; i < n_admissible_blocks; ++i ){
  getAdmissibleBlock( i, localBlock );
  ACAMatrix.addAdmissibleBlock( i, localBlock );
}
```

Experiment setting

- Full (ACA) assembly tested on a mesh with 20.480 (81.920) surface elements.
- 4 quadrature points in each dimension (256 per simplex) to utilize SIMD registers.
- ACA settings
  - maximal number of elements in clusters: 500,
  - preallocation: 10 %.

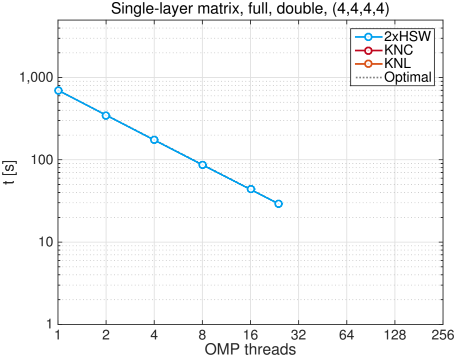
- Assembly performed on single nodes
  - Salomon - 2 x **Xeon 2680v3**, AVX2, 2x12 cores, 2.5 GHz, 128 GB RAM,
  - Salomon - **Xeon Phi 7120P**, IMCI, 61 cores, 1.238 GHz, 16 GB RAM,
  - Endeavor - **Xeon Phi 7210**, AVX-512, 64 cores, 1.3 GHz, 16 + 96 GB RAM.

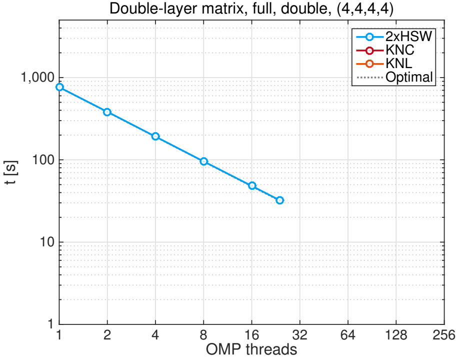
Experiment setting

- Full (ACA) assembly tested on a mesh with 20.480 (81.920) surface elements.
- 4 quadrature points in each dimension (256 per simplex) to utilize SIMD registers.
- ACA settings
  - maximal number of elements in clusters: 500,
  - preallocation: 10 %.

- Assembly performed on single nodes
  - Salomon - 2 x **Xeon 2680v3**, AVX2, 2x12 cores, 2.5 GHz, 128 GB RAM,
  - Salomon - **Xeon Phi 7120P**, IMCI, 61 cores, 1.238 GHz, 16 GB RAM,
  - Endeavor - **Xeon Phi 7210**, AVX-512, 64 cores, 1.3 GHz, 16 + 96 GB RAM.

OMP scalability, full assembly, 256 points per simplex

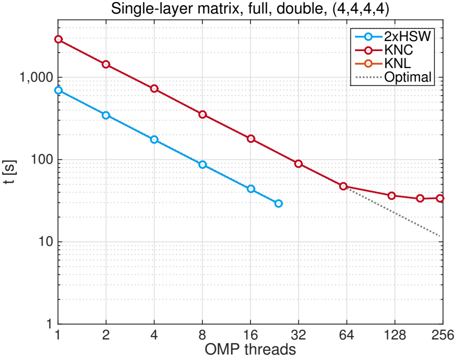


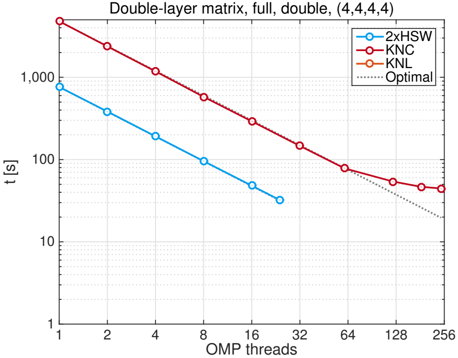


Xeon 2680v3

matrix	2	4	8	16	24
V_h	2.01	4.04	8.07	16.09	24.07
K_h	1.99	3.98	7.97	15.83	23.68

OMP scalability, full assembly, 256 points per simplex

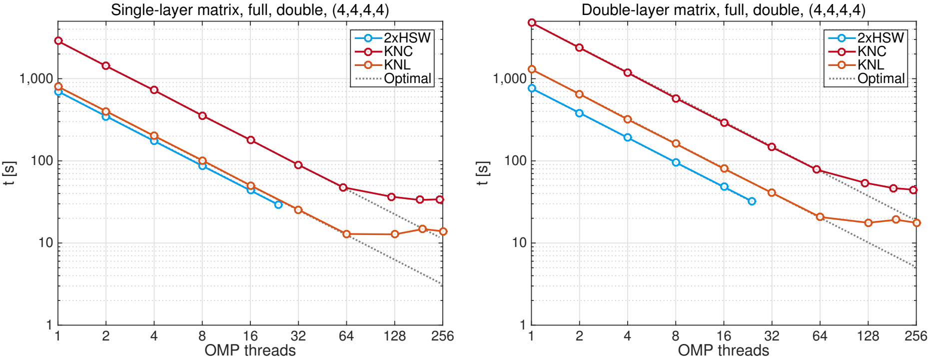




Xeon Phi 7120P

matrix	2	4	8	16	32	61	122	183	244
V_h	2.01	3.98	8.06	15.96	32.14	60.23	78.33	85.90	84.29
K_h	2.02	4.06	8.28	16.42	32.72	61.38	88.54	102.71	107.53

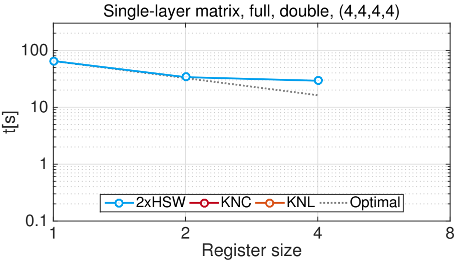
OMP scalability, full assembly, 256 points per simplex

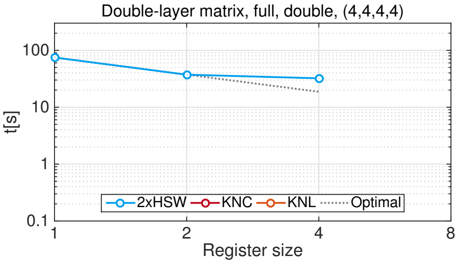


Xeon Phi 7210

matrix	2	4	8	16	32	64	128	192	256
V_h	1.99	3.99	7.90	15.93	31.48	62.26	62.89	54.22	57.65
K_h	2.00	4.07	8.07	16.20	31.89	62.76	73.64	67.76	73.64

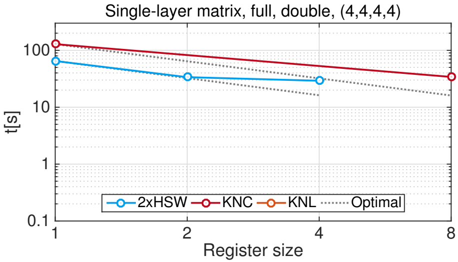
SIMD scalability, full assembly, 256 points per simplex

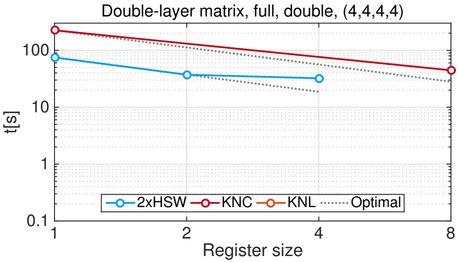




architecture	threads	matrix	SSE4.2	AVX2	IMCI	AVX-512
Xeon E2680v3	24	V_h	1.92	2.23	—	—
		K_h	2.01	2.32	—	—
Xeon Phi 7120P	244	V_h	—	—	3.77	—
		K_h	—	—	5.05	—
Xeon Phi 7210	128	V_h	2.45	4.95	—	9.94
		K_h	2.01	4.14	—	7.69

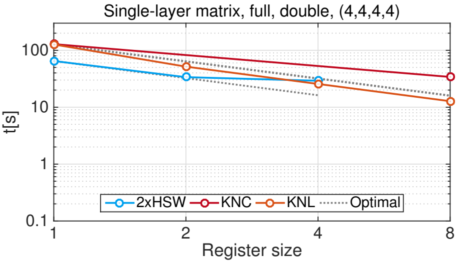
SIMD scalability, full assembly, 256 points per simplex

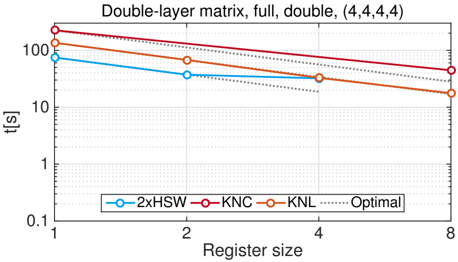




architecture	threads	matrix	SSE4.2	AVX2	IMCI	AVX-512
Xeon E2680v3	24	V_h	1.92	2.23	—	—
		K_h	2.01	2.32	—	—
Xeon Phi 7120P	244	V_h	—	—	3.77	—
		K_h	—	—	5.05	—
Xeon Phi 7210	128	V_h	2.45	4.95	—	9.94
		K_h	2.01	4.14	—	7.69

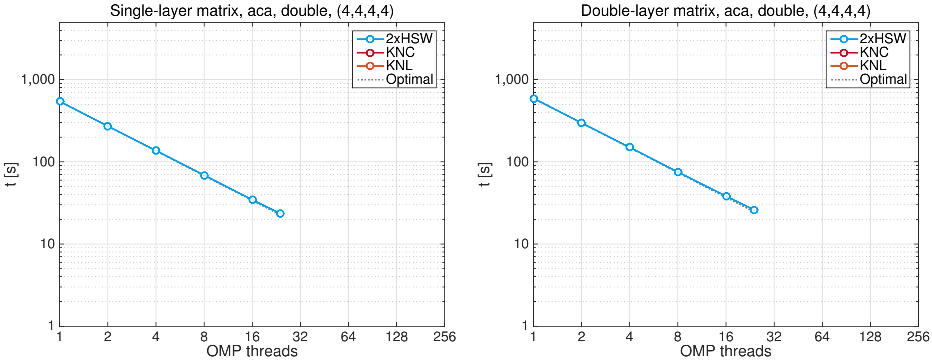
SIMD scalability, full assembly, 256 points per simplex





architecture	threads	matrix	SSE4.2	AVX2	IMCI	AVX-512
Xeon E2680v3	24	V_h	1.92	2.23	—	—
		K_h	2.01	2.32	—	—
Xeon Phi 7120P	244	V_h	—	—	3.77	—
		K_h	—	—	5.05	—
Xeon Phi 7210	128	V_h	2.45	4.95	—	9.94
		K_h	2.01	4.14	—	7.69

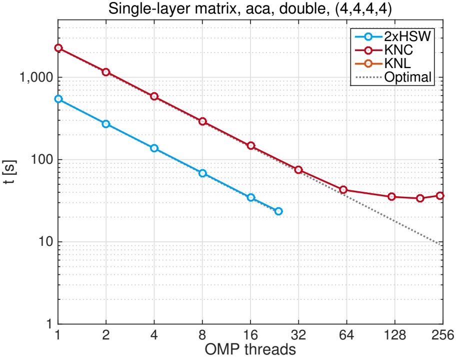
OMP scalability, ACA assembly, 256 points per simplex

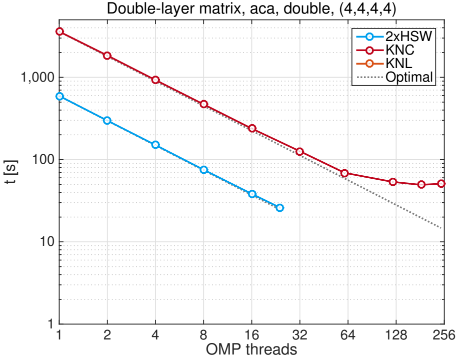


Xeon 2680v3

matrix	2	4	8	16	24
V_h	2.00	3.97	7.89	15.74	23.13
K_h	1.99	3.95	7.89	15.49	22.75

OMP scalability, ACA assembly, 256 points per simplex

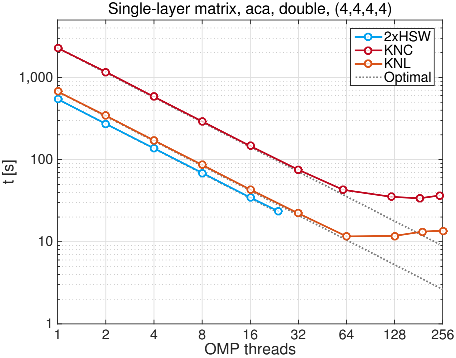


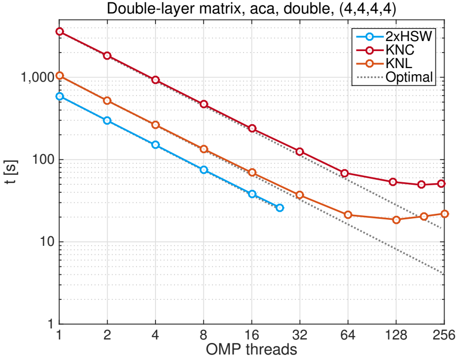


Xeon Phi 7120P

matrix	2	4	8	16	32	61	122	183	244
V_h	1.95	3.92	7.80	15.57	30.32	53.16	64.69	67.44	62.39
K_h	1.95	3.88	7.67	15.10	28.49	52.17	67.20	72.94	70.66

OMP scalability, ACA assembly, 256 points per simplex

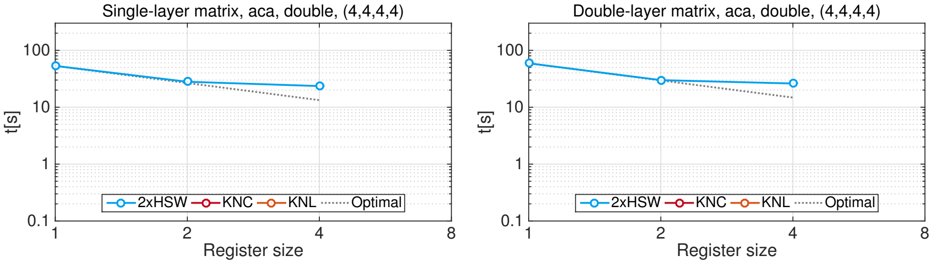




Xeon Phi 7210

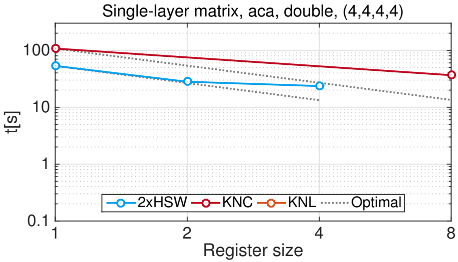
matrix	2	4	8	16	32	64	128	192	256
V_h	1.97	3.93	7.80	15.57	30.36	57.92	57.08	50.57	49.34
K_h	2.01	3.95	7.78	15.03	28.34	49.17	55.92	51.02	47.30

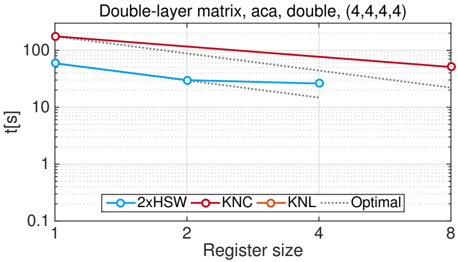
SIMD scalability, ACA assembly, 256 points per simplex



architecture	threads	matrix	SSE4.2	AVX2	IMCI	AVX-512
Xeon E2680v3	24	V_h	1.90	2.26	—	—
		K_h	1.98	2.27	—	—
Xeon Phi 7120P	244	V_h	—	—	2.94	—
		K_h	—	—	3.47	—
Xeon Phi 7210	128	V_h	2.33	4.45	—	8.06
		K_h	1.96	3.71	—	5.43

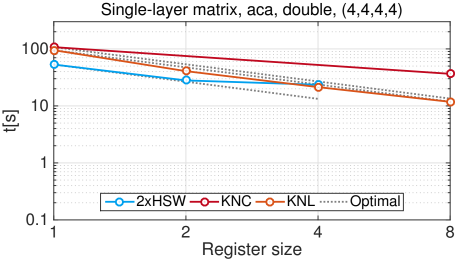
SIMD scalability, ACA assembly, 256 points per simplex

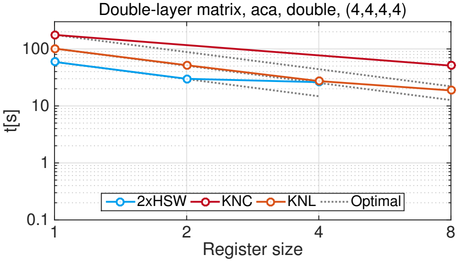




architecture	threads	matrix	SSE4.2	AVX2	IMCI	AVX-512
Xeon E2680v3	24	V_h	1.90	2.26	—	—
		K_h	1.98	2.27	—	—
Xeon Phi 7120P	244	V_h	—	—	2.94	—
		K_h	—	—	3.47	—
Xeon Phi 7210	128	V_h	2.33	4.45	—	8.06
		K_h	1.96	3.71	—	5.43

SIMD scalability, ACA assembly, 256 points per simplex





architecture	threads	matrix	SSE4.2	AVX2	IMCI	AVX-512
Xeon E2680v3	24	V_h	1.90	2.26	—	—
		K_h	1.98	2.27	—	—
Xeon Phi 7120P	244	V_h	—	—	2.94	—
		K_h	—	—	3.47	—
Xeon Phi 7210	128	V_h	2.33	4.45	—	8.06
		K_h	1.96	3.71	—	5.43

Conclusion

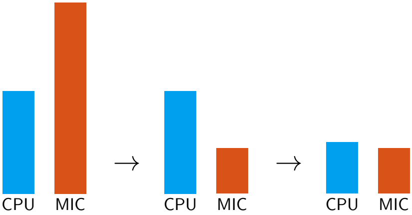
- KNL vs. HSW speedup
  - Full – V_h 2.29, K_h 1.82,
  - ACA – V_h 2.64, K_h 1.40.

- What we learned
  - SIMD processing becoming more efficient with KNL,
  - multi-core code benefits from many-core optimizations.

- Work in progress
  - object-oriented offload to Xeon Phi (full, ACA),
  - load balancing for offload mode (full, ACA),
  - massively parallel BETI with ESPRESO library (MPI, OpenMP, SIMD, offload).

Conclusion

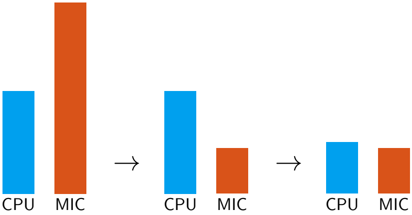
- KNL vs. HSW speedup
  - Full – V_h 2.29, K_h 1.82,
  - ACA – V_h 2.64, K_h 1.40.
- What we learned
  - SIMD processing becoming more efficient with KNL,
  - multi-core code benefits from many-core optimizations.



- Work in progress
  - object-oriented offload to Xeon Phi (full, ACA),
  - load balancing for offload mode (full, ACA),
  - massively parallel BETI with ESPRESO library (MPI, OpenMP, SIMD, offload).

Conclusion

- KNL vs. HSW speedup
  - Full – V_h 2.29, K_h 1.82,
  - ACA – V_h 2.64, K_h 1.40.
- What we learned
  - SIMD processing becoming more efficient with KNL,
  - multi-core code benefits from many-core optimizations.



- Work in progress
  - object-oriented offload to Xeon Phi (full, ACA),
  - load balancing for offload mode (full, ACA),
  - massively parallel BETI with ESPRESO library (MPI, OpenMP, SIMD, offload).

References

Zapletal, J.; Merta, M.; Malý, L.
Boundary element quadrature schemes for multi- and many-core architectures.
Computers and Mathematics with Applications (accepted).

Merta, M.; Zapletal, J.; Jaroš, J.
Many core acceleration of the boundary element method.
High Performance Computing in Science and Engineering: Second International Conference, HPCSE 2015, Soláň, Czech Republic, May 25-28, 2015, Revised Selected Papers, Springer International Publishing 14:116–125, 2016.

Preprints available at **researchgate.com**.

Thank you

References

Zapletal, J.; Merta, M.; Malý, L.
Boundary element quadrature schemes for multi- and many-core architectures.
Computers and Mathematics with Applications (accepted).

Merta, M.; Zapletal, J.; Jaroš, J.
Many core acceleration of the boundary element method.
High Performance Computing in Science and Engineering: Second International Conference, HPCSE 2015, Soláň, Czech Republic, May 25-28, 2015, Revised Selected Papers, Springer International Publishing 14:116–125, 2016.

Preprints available at **researchgate.com**.

Thank you