Optimizing the ESPRESO solver based on the hybrid FETI method for MIC architectures

Vít Vondrák

Lubomír Říha, Michal Merta, Ondřej Meca, Alexandros Markopoulos, Tomáš Brzobohatý
Outline

• Introduction to IT4Innovations computing infrastructure
• Total FETI and Hybrid Total FETI method
  – Intel Parallel Computing center at IT4Innovations (ESPRESO solver)
  – Hybrid FETI method
  – Acceleration of FETI method on Intel MIC
• Performance and scalability of ESPRESO
• BEM4I – Boundary element method code on Intel MIC
ITU4Innovations infrastructure history

1. **Anselm**
   94 TFLOPs system
   June 2013

2. **IT4I building**
   July 2014

3. **Salomon**
   2 PFLOPs system
   July 2015
Anselm in numbers

- 209 compute nodes
- 3344 Intel Sandy bridge cores
- 15136 GB RAM (64, 96, 512)
- 24 nVidia Tesla K20
- 4 Intel Xeon Phi 5110P (240 cores)

- Rpeak 94TFlop/s \( (94 \times 10^{12} \text{ flops per sec}) \)
- Rmax 73TFlop/s (LINPACK)
Salomon in numbers

- 1008 compute nodes
- 24192 Intel Haswell cores
- 129024 GB RAM (128)
- 864 Intel Xeon Phi 7120P (52704 cores)

- $\text{R}_{\text{peak}}$ 2PFlop/s (2*10^{15} flops per sec)
- $\text{R}_{\text{max}}$ 1.5Flop/s (LINPACK)

- #55 in top500.org (July 2015)
- #20 in Europe (June 2015)
- (#1 Intel Xeon Phi in Europe)
Intel Xeon Phi coprocessors in Salomon

**Xeon Phi 7120P**
- x86 architecture, **1.2TF**
- 864x Intel Xeon Phi 7120P
- 61(244) cores, 512 bit FMA
- 16 GB RAM
Intel KNC Architecture

Up to 61 Cores, 244 Threads
512-bit SIMD instructions
>1 TFLOPS DP-F.P. peak

Up to 16GB GDDR5 Memory
- 352 GB/s peak, but ~170 GB/s measured

PCle 2.0 x16 - 5.0 GT/s, 16-bit Data Cache
- L1 32KB/core
- L2 512KB/core, 30.5 MB/chip

Up to 300W TDP (card)

Linux* operating system
- IP addressable - coprocessor becomes a network node
- Common x86/IA
- Programming Models and SW-Tools
Users of IT4Innovations infrastructure

142 mil corehours distributed
584 users in 234 projects
avg. util. 70%, max util. 98%

Allocated IT4I resources since 6/2013

- Ostrava: 38%
- Brno: 29%
- Praha: 22%
- Other EU: 4%
- Other CR: 7%
- Avg. util.: 70%
- Max util.: 98%
Intel® Parallel Computing Center

- **WP1: Development of highly parallel algorithms and libraries**
  - Algorithm development & implementation - ESPRESO
  - Algorithm optimization
- **WP2: Development and support of HPC community codes**
  - Development of the API
  - Plug-ins for selected community codes – OpenFOAM, ELMER
Various input format including API
Mesh processing and Matrix Assemblers
Multi-level domain decomposition method
Support for modern multi and many-core accelerators
Total FETI method

**Total FETI**

- Non-overlapping domain decomposition method
- Mutual continuity of primal variables between neighboring subdomains is enforced by dual variables, i.e., Lagrange multipliers obtained iteratively by the Krylov subspace methods
**Projected Conjugate Gradient method**

\[
g_{k+1} = g_k + \alpha_k F p_k \quad g_{k+1}^{proj} = Pg_{k+1}
\]

\[
P = I - G^T (G G^T)^{-1} G
\]

\[
\min \frac{1}{2} u^T K u - u^T f \quad \text{s.t. } B u = c
\]

\[
\min \frac{1}{2} \lambda^T F \lambda - \lambda^T d \quad \text{s.t. } G \lambda = 0
\]

\[
F = B K^+ B^T, \quad G^T = -B R,
\]

\[
d = B K^+ f - c
\]
Total FETI: Weak scalability

24³ - domain size
4³ subdomains per node – processed in parallel on 24 cores using Cilk++
Test ran on: 1, 8, 27, 64, 125, 216 and 343 nodes (1³ ... 7³) – each 24 cores
12^3 - domain size
9^3 subdomains per node – processed in parallel on 24 cores using Cilk++
Test ran on: 1, 8, 27 nodes – each 24 cores
Total FETI and Hybrid Total FETI Methods

Hybrid Total FETI

- Non-overlapping domain decomposition method
- Subdomains grouped into non-overlapping clusters
- Mutual continuity of primal variables between neighboring subdomains is enforced by dual variables, i.e., Lagrange multipliers obtained iteratively by the Krylov subspace methods.
Hybrid Total FETI solver – Why is it scalable?

Total FETI (2D case)
Problem decomposed into 4 subdomains generates **coarse problem matrix** \((GG^T)\) with dimension:

\[ 3 \times (\text{number of SUBDOMAINS}) = 12 \]

Modified system has ‘only’ 6 independent rigid motions.

Hybrid Total FETI (2D case)
Beam decomposed into 2 clusters (each consists of \(N\) subdomains) generates **coarse problem matrix** \((GG^T)\) with dimension

\[ 3 \times (\text{number of CLUSTERS}) = 6 \]

Number of clusters = number of nodes
**HTFETI: Weak scalability**

12^3 - domain size
9^3 subdomains per cluster - HFETI corners in corners – no corners on edges
Test ran on: 1, 8, 27, 64, 125, 216 and 343 nodes (1^3 ... 7^3) – each 24 cores
Main blocks of the Hybrid Total FETI solver

Iterative solver step *(CG solver runtime)*
- the most time consuming part
- Runtime depends on number of iterations = how well the problem is conditioned
- Essential kernels will be discussed later

Preprocessing step – all operations executed one time
Coarse problem matrix assembling and calculation of its inverse *(FETI preprocessing)*
- Assembling – nearest neighbor MPI communication required
- Inverse matrix calculation – 1 MPI broadcast – local factorization and solve (PARDISO)
- Additional preprocessing inside clusters *(HTFETI preprocessing)*
  - Several local factorizations and solves using PARDISO
- Stiffness matrix factorization *(K factorization)*
  - Local factorizations and solve using PARDISO
Computing kernels of FETI method

Projected Conjugate Gradient in FETI

1: \( r_0 := b - A x_0; \quad u_0 := M^{-1} r_0; \quad p_0 := u_0 \)
2: for \( i = 0, \ldots, m - 1 \) do
3: \( s := A p_i \)
4: \( \alpha := \langle r_i, u_i \rangle / \langle s, p_i \rangle \)
5: \( x_{i+1} := x_i + \alpha p_i \)
6: \( r_{i+1} := r_i - \alpha s \)
7: \( u_{i+1} := M^{-1} r_{i+1} \)
8: \( \beta := \langle r_{i+1}, u_{i+1} \rangle / \langle r_i, u_i \rangle \)
9: \( p_{i+1} := u_{i+1} + \beta p_i \)
10: end for

90 – 95% of runtime spent in \( A p_i \)

Pre-processing – K factorization

1.) \( x = B^T_1 \cdot \lambda \) - SpMV
2.) \( y = K^{-1} \cdot x \) - solve on CPU
3.) \( \lambda = B_1 \cdot y \) - SpMV
4.) stencil data exchange in \( \lambda \)
   - MPI – Send and Recv
   - OpenMP – shared mem. vec

Sparse Matrix-Vector product

- Only communication with neighbors
- Good scaling

Dot-product

- Global communication
- Scales as \( \log(P) \)
- Scalar vector multiplication, vector-vector addition
- No communication

IT4 Innovations
national supercomputing center
How to Accelerate FETI methods with Xeon Phi

Approach 1 – Using Sparse Matrices

Projected Conjugate Gradient in FETI

1: \( r_0 := b - A x_0; u_0 := M^{-1} r_0; p_0 := u_0 \)
2: \textbf{for} \( i = 0, \ldots, m - 1 \) \textbf{do}
3: \[ s := A p_i \]
4: \[ \alpha := \langle r_i, u_i \rangle / \langle s, p_i \rangle \]
5: \[ x_{i+1} := x_i + \alpha p_i \]
6: \[ r_{i+1} := r_i - \alpha s \]
7: \[ u_{i+1} := M^{-1} r_{i+1} \]
8: \[ \beta := \langle r_{i+1}, u_{i+1} \rangle / \langle r_i, u_i \rangle \]
9: \[ p_{i+1} := u_{i+1} + \beta p_i \]
10: \textbf{end for}

90 – 95% of runtime spent in \( A p_i \)

Pre-processing - \( K \rightarrow \text{MIC} - K \) factorization on MIC
1.) \( x = B_1^T \cdot \lambda \) - SpMV on CPU
2.) \( x \rightarrow \text{MIC} \) - PCIe transfer from CPU
3.) \( y = K^{-1} \cdot x \) - solve on MIC
4.) \( y \leftarrow \text{MIC} \) - PCIe transfer to CPU
5.) \( \lambda = B_1 \cdot y \) - SpMV on CPU
6.) stencil data exchange in \( \lambda \)
   - MPI – Send and Recv
   - OpenMP – shared mem. vec
How to Accelerate FETI methods with Xeon Phi

Approach 1 – Using Sparse Matrices

<table>
<thead>
<tr>
<th></th>
<th>Haswell 1x E5-2680v3 CPU</th>
<th>MIC 1x Intel Xeon Phi 7120p accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12 cores</td>
<td>60 threads</td>
</tr>
<tr>
<td>dim.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2187 x 2187 – 2800 domains</td>
<td>Fact. [s]</td>
<td>Solve [s]</td>
</tr>
<tr>
<td></td>
<td>6.9</td>
<td>17.0</td>
</tr>
<tr>
<td>6591 x 6591 – 700 domains</td>
<td>7.2</td>
<td>18.7</td>
</tr>
<tr>
<td>10125 x 10125 – 360 domains</td>
<td>6.9</td>
<td>17.4</td>
</tr>
<tr>
<td>12288 x 12288 – 300 domains</td>
<td>7.8</td>
<td>18.5</td>
</tr>
</tbody>
</table>

Factorization (preprocessing) - **7120p Xeon Phi is approximately 3-4x slower than CPU**

Solver (iterative solver) - **7120p Xeon Phi is as fast as CPU**

With this approach preprocessing time remains the same, but iterative solver processing time is reduced by 50% by Intel Xeon Phi.
How to Accelerate FETI methods with Xeon Phi

Approach 2 – Using Dense Matrices (Schur Complement)

Projected Conjugate Gradient in FETI

1: \( r_0 := b - Ax_0; u_0 := M^{-1}r_0; p_0 := u_0 \)
2: \( \text{for } i = 0, \ldots, m - 1 \text{ do} \)
3: \( s := Ap_i \)
4: \( \alpha := \langle r_i, u_i \rangle / \langle s, p_i \rangle \)
5: \( x_{i+1} := x_i + \alpha p_i \)
6: \( r_{i+1} := r_i - \alpha s \)
7: \( u_{i+1} := M^{-1}r_{i+1} \)
8: \( \beta := \langle r_{i+1}, u_{i+1} \rangle / \langle r_i, u_i \rangle \)
9: \( p_{i+1} := u_{i+1} + \beta p_i \)
10: \text{end for} \)

90 – 95% of runtime spent in \( Ap_i \)

Pre-processing: \( S_c = B_1 K^{-1} B_1^T \rightarrow \text{MIC} \)
1.) \( \lambda \rightarrow \text{MIC} \) - PCIe transfer from CPU
2.) \( \lambda = S_c \cdot \lambda \) - DGEMV, DSYMV on MIC
3.) \( \lambda \leftarrow \text{MIC} \) - PCIe transfer to CPU
4.) stencil data exchange in \( \lambda \)
   - MPI – Send andRecv
   - OpenMP – shared mem. vec

Requires algorithmic changes in the FETI solver in both preprocessing and iterative solver steps

Key features this approach:
- Increases preprocessing time – Schur Complement (SC) computation for each subdomain
- Reduce iterative solver time – single iteration time is reduced
## Schur Complement Computation on CPU and Xeon Phi

### Comparison of SC computation using PARDISO SC and MKL

<table>
<thead>
<tr>
<th>Domain size x number of domains</th>
<th>Haswell 1x E5-2680v3 CPU</th>
<th>MIC 1x Intel Xeon Phi 7120p accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>2187 x 2187 - 1500 subdomains</td>
<td>26.7</td>
<td>70.4</td>
</tr>
<tr>
<td>6591 x 6591 - 250 subdomains</td>
<td>28.5</td>
<td>80.5</td>
</tr>
<tr>
<td>12288 x 10125 - 60 subdomains</td>
<td>29.5</td>
<td>59.5</td>
</tr>
</tbody>
</table>

### Key features of this approach:
- Increases preprocessing time – Schur Complement (SC) computation for each subdomain
- Reduce iterative solver time – single iteration time is reduced

**Number of subdomains:**
- CPU (24 th.): 1331
- Xeon Phi: 512

**Stiffness matrix sizes:**
- CPU: 2187 x 2187
- Xeon Phi: 2187 x 2187

**Symmetric:**
- CPU: 16.2 GB
- Xeon Phi: 7.6 GB

**Number of iterations:**
- CPU: 500
- Xeon Phi: 500

**Speedup:**
- 2.5 times
ESPRESO on Salomon

2,016 Intel Xeon E5-2680v3, 2.5GHz, 12cores
864 Intel Xeon Phi 7120P, 61cores, 16GB RAM
ESPRESO Problem Generator

Massively parallel Benchmark Generator
- designed to perform large scale tests – tested up to 120 billion unknowns
- problem with all matrix objects generated in seconds

Model problems
- Cube
- Sphere

Physics
- Laplace equation
- Linear Elasticity
CPU vs MIC solver

7.5 – 2912 million DOF Hybrid FETI CG Solver Runtime
Laplace – Single Iteration Time
IT4Innovations Salomon Supercomputer

- CPU - PARDISO - Lumped prec
- CPU - SC - NO prec
- MIC - SC - NO prec

speedUp

- (11.3)
- 2.3

stopping criteria: 1e-4
subdomain size: 4096 DOF
cluster size: 2197 subdomains
cluster size: 7.5 million DOF
CPU vs MIC solver

7.5 – 2912 million DOF Hybrid FETI CG Solver Runtime

Laplace – Single Iteration Time

IT4Innovations Salomon Supercomputer

speedUp

- 7.8
- 1.9

Processing time [s]

CPU - PARDISO - Lumped prec

CPU - SC - Lumped prec

MIC - SC - Lumped prec

stopping criteria: 1e-4
subdomain size: 4096 DOF
cluster size: 2197 subdomains
cluster size: 7.5 million DOF

Number of compute nodes [-]
CPU vs MIC solver

7.5 – 2912 million DOF Hybrid FETI CG Solver Runtime

Laplace – Single Iteration Time

IT4Innovations Salomon Supercomputer

Stopping criteria: 1e-4
Subdomain size: 4096 DOF
Cluster size: 2197 subdomains
Cluster size: 7.5 million DOF
CPU vs MIC solver

7.5 – 2912 million DOF Hybrid FETI CG Solver Runtime
Laplace – CG Solver Runtime w/o Precon.

IT4Innovations Salomon Supercomputer

speedUp

CPU - PARDISO - Lumped prec → CPU - SC - NO prec → MIC - SC - NO prec

processing time [s]

stopping criteria: 1e-4
subdomain size: 4096 DOF
cluster size: 2197 subdomains
cluster size: 7.5 million DOF

CPU vs MIC solver
CPU vs MIC solver

7.5 – 2912 million DOF Hybrid FETI CG Solver Runtime
Laplace – CG Solver Runtime w. Lumped Precon.

IT4Innovations Salomon Supercomputer

speedUp
7.8
1.9

Processing time [s] vs Number of compute nodes [-]

stopping criteria: 1e-4
subdomain size: 4096 DOF
cluster size: 2197 subdomains
cluster size: 7.5 million DOF

CPU vs MIC solver
CPU vs MIC solver

7.5 – 2912 million DOF Hybrid FETI CG Solver Runtime
Laplace – CG Solver Runtime w/o Preprocessing

IT4Innovations Salomon Supercomputer

- CPU - PARDISO - Lumped prec
- CPU - SC - Lumped prec
- CPU - SC - NO prec
- MIC - SC - Lumped prec
- MIC - SC - NO prec

Processing time [s]

- Stopping criteria: 1e-4
- Subdomain size: 4096 DOF
- Cluster size: 2197 subdomains
- Cluster size: 7.5 million DOF

Number of compute nodes [-]
ESPRESO on TITAN

18,688  AMD Opteron 6274 16-core CPUs
18,688  Nvidia Tesla K20X GPUs

2.7 million core hours dedicated to:
• scalability optimization of ESPRESO
• optimization of GPU accelerated version for large scale problems

TITAN 3rd in TOP500 LIST
Strong Scalability Test

20 billion DOF on up to 17,576 Compute Nodes (281,216 cores)
Heat transfer (Laplace equation)

ORNL Titan 2\textsuperscript{nd} in TOP500 LIST

![Solver runtime vs. Number of compute nodes graph]

- Linear
- Real

- Solver runtime [s]
- Number of compute nodes
Weak Scalability Test

Up to 124 billion DOF on 17576 Compute Nodes (281 216 cores)
Heat transfer (Laplace equation)

ORNL Titan 2nd in TOP500 LIST

Solver Time [s]

<table>
<thead>
<tr>
<th>Problem size [billion DOF]</th>
<th>Solver Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.53</td>
<td>47</td>
</tr>
<tr>
<td>216</td>
<td>47</td>
</tr>
<tr>
<td>3.62</td>
<td>47</td>
</tr>
<tr>
<td>512</td>
<td>47</td>
</tr>
<tr>
<td>7.07</td>
<td>47</td>
</tr>
<tr>
<td>1000</td>
<td>47</td>
</tr>
<tr>
<td>12.2</td>
<td>47</td>
</tr>
<tr>
<td>1728</td>
<td>47</td>
</tr>
<tr>
<td>19.4</td>
<td>47</td>
</tr>
<tr>
<td>2744</td>
<td>47</td>
</tr>
<tr>
<td>28.9</td>
<td>47</td>
</tr>
<tr>
<td>4096</td>
<td>47</td>
</tr>
<tr>
<td>41.2</td>
<td>47</td>
</tr>
<tr>
<td>5832</td>
<td>47</td>
</tr>
<tr>
<td>56.5</td>
<td>47</td>
</tr>
<tr>
<td>8000</td>
<td>47</td>
</tr>
<tr>
<td>75.2</td>
<td>47</td>
</tr>
<tr>
<td>10648</td>
<td>47</td>
</tr>
<tr>
<td>97.6</td>
<td>47</td>
</tr>
<tr>
<td>13824</td>
<td>47</td>
</tr>
<tr>
<td>124</td>
<td>47</td>
</tr>
</tbody>
</table>

Problem size [billion DOF]
Number of compute nodes [-]
Strong Scalability Test

300 million unknown - ANSYS Workbench real world problem
Linear elasticity

IT4Innovations – SALOMON Supercomputer
Strong Scalability Test

300 million unknown - ANSYS Workbench
Linear elasticity

IT4Innovations – SALOMON Supercomputer

Total solver runtime [s]

MPI processes/CPU cores

Linear

Real
Boundary element library: BEM4I

- Developed at IT4Innovations NSC
- Reduces problem to the boundary of a computational domain
- Suitable mainly for problems on unbounded domains or shape optimization
- Heat transfer, wave scattering, linear elasticity
Boundary element library: BEM4I

- Templated C++ library
- SIMD vectorization (using Intel's pragmas or Vc library)
- OpenMP and MPI parallelization
- Intel Xeon Phi acceleration

BEM4I

Matrix Assembler
  Laplace, Lamé, Helmholtz, Wave
  Dense or sparsified

Iterative solvers
  BETI (interf. ESPRESO)

Export
  ParaView

CPU

MIC
Discretization

- Galerkin method for discretization of the boundary integral equation
- Matrix formulation

\[
\begin{aligned}
\text{Find } & t_h \in \mathbb{R}^N \text{ such that} \\
V t_h = (\frac{1}{2} M_h + K_h) y_h.
\end{aligned}
\]

- System matrices (single layer and double layer matrix)

\[
\begin{aligned}
V_h[i, j] &:= \frac{1}{4\pi} \int_{\Gamma_i} \int_{\Gamma_j} \frac{1}{\| x - y \|} \, ds_y \, ds_x \\
K_h[k, l] &:= \frac{1}{4\pi} \int_{\Gamma_k} \int_{\Gamma_l} \frac{(x - y, n_y)}{\| x - y \|^3} \varphi(t) \, ds_y \, ds_x
\end{aligned}
\]

- System matrix assembly - quadratic complexity
  - Computationally most demanding part of BEM
  - Parallelized by OpenMP, MPI
  - Accelerated by Intel Xeon Phi coprocessor
  - Possible utilization of Fast BEM methods (sparsification)
Acceleration of the system matrix assembly

- Matrix split into parts for host CPUs and coprocessors
  - Load balanced according to theoretical performance of host CPUs and MICs
- Coprocessor parts further split into smaller submatrices
  - To fit into coprocessor memory
  - To overlap communication by computation (double buffering)
- Computation accelerated using offload mode of the coprocessor
- On the coprocessor the code is parallelized using ordinary OpenMP pragmas
- Vectorization of the code (e.g. using #pragma simd) and scalability up to hundreds of threads necessary to obtain good performance on the coprocessor

\[ V_h[\ell,j] := \int_{\tau_{\ell}} \int_{\tau_j} v(x,y) \, ds_y \, ds_x \]
Acceleration of the system matrix assembly

Laplace equation

- 81920 surface elements
- Maximum speedup using two cards approx. 2.5
Acceleration of the system matrix assembly

Laplace equation

- 81920 surface elements
- Maximum speedup using two cards approx. 2.5

![Graph showing system matrix assembly time for single precision](image-url)
BETI for linear elasticity

• Accelerating BETI (Boundary Element Tearing and Interconnecting) domain decomposition method for the linear elasticity problems
• Interface to the ESPRESO domain decomposition library
• BEM4I generates the local Dirichlet-to-Neumann map for each subdomain (Steklov-Poincaré operator)

\[ S_h = (1/2M_h + K_h^\top) V_h^{-1} (1/2M_h + K_h) \]
BETI for linear elasticity

- 20480 surface elements
- Maximum speedup using two cards
  - V, Vlap, Klap (in red): 2.5
  - Total: 1.8
BETI for linear elasticity

Non-accelerated BETI up to 64 nodes/57 mil. surface unknowns

- Currently working on reducing the time for Steklov-Poincaré assembly using accelerators
Conclusions

- Hybrid FETI implementation shows very high scalability
  - Numerical and parallel
  - Strong and weak
- Many core architectures bring higher performance
  - But this is not for free
  - Only certain parts of the code may be accelerated
- Codes with full matrices like BEM4I benefit from MIC architecture