ABSTRACT

This paper describes the Hybrid Total FETI (HTFETI) method and its parallel implementation in the ESPRESO library. HTFETI is a variant of the FETI type domain decomposition method in which a small number of neighboring subdomains is aggregated into clusters. This can be also viewed as a multilevel decomposition approach which results into a smaller coarse problem – the main scalability bottleneck of the FETI and FETI-DP methods.

The efficiency of our implementation which employs hybrid parallelization in the form of MPI and Cilk++ is evaluated using both weak and strong scalability tests. The weak scalability of the solver is shown on the 3 dimensional linear elasticity problem of a size up to 30 billion of Degrees Of Freedom (DOF) executed on 4096 compute nodes. The strong scalability is evaluated on the problem of size 2.6 billion DOF scaled from 1000 to 4913 compute nodes. The results show the super-linear scaling of the single iteration time and linear scalability of the solver runtime. The latter combines both numerical and parallel scalability and shows overall HTFETI solver performance. The large scale tests use our own parallel synthetics benchmark generator that is also described in the paper.

The last set of results shows that HTFETI is very efficient for problems of size up 1.7 billion DOF and provide better time to solution when compared to TFETI method.

CCS Concepts

- Mathematics of computing → Solvers;

Keywords

Hybrid Total FETI, HTFETI, Linear elasticity, hybrid parallelization, massively parallel

1. INTRODUCTION

For more than two decades, the Finite Element Tearing and Interconnecting method (FETI, for more detail see, e.g., in [5], [7]) has been successfully used in the engineering community, primarily for very large problems arising from the discretization of partial differential equations. In such an approach the original structure is artificially decomposed into several non-overlapping subdomains. Mutual continuity of primal variables between neighboring subdomains is enforced afterwards by dual variables, i.e., Lagrange multipliers (LM). They are usually obtained iteratively by some of the Krylov subspace methods, then the primal solution is evaluated locally on each subdomain. The FETI method is also applicable for contact problems with inequality constraints (see, e.g., [3],[4]).

In 2006 Dostál et al. [2] introduced a new variant of an algorithm called Total FETI (or TFETI) in which Dirichlet boundary condition is enforced also by LM. Please note, there is no difference, if the hybrid variant stems from FETI or TFETI. The impact is identical in both cases. This pa-
The comparison of HTFETI and TFETI methods is shown in Figure 1. Using TFETI with large number of DOF per subdomain is an example of how to solve large problems using this technique. The figure shows that its scalability is not ideal, but is acceptable up to 3 billion DOF. However, the solution time in this range is still shorter than it would be with the use of the HTFETI technique. The figure clearly shows that for problems greater than 3 billion DOF, the HTFETI Approach A is better.

However, the HTFETI can be configured to the other extreme in which small number of subdomains per cluster (here 216) and very small number of DOF per subdomain (here 1536 DOF) are used. This approach is significantly faster than TFETI with large number of DOF per subdomain. Figure also shows that TFETI can be configured with smaller number of DOF per subdomain (here 20,577) to reduce processing time. However, this option is suitable only for very small problems as its scalability is unsatisfactory.

In sum, the figure shows that HTFETI provides better solution time for "smaller problems" up to 3 billion DOF and better scalability for large problems.

In the next section, the theory behind the HTFETI is described, followed by the description of the parallel implementation in the ESPRESO library.

### 2. HYBRID TOTAL FETI METHOD

The FETI method, widely known for more than two decades, is an efficient tool for solving large-scale problems in structural mechanics via Krylov subspace methods adapted to parallel machines. Although this paper is focused on the HTFETI, this section will first introduce the original FETI method on a simple cantilever beam (see Figure 2), followed by an introduction of its hybrid variant.

#### 2.1 FETI

In the simple engineering problem depicted in Figure 2, the aim is to get deformation of the structure. A cantilever beam, fixed on the left side and loaded by the pressure \( p_y \) at the top, is discretized with FEM [11]. The number of nodes is \( n_n \), and number of elements is \( n_e \). Above the mesh global stiffness matrix \( K_g \) and global RHS \( f_g \) (nodal forces) are assembled, whereas both objects are included into nodal equilibrium equation

\[
K_g u_g = f_g.
\]

To get the vector \( u_g \) (nodal displacements), Dirichlet boundary condition has to be taken into account due to a singular matrix \( K_g \) since it has a non-empty null-space. Then the linear system can be solved by an iterative or direct solver. Clearly, this undecomposed approach has its own limitations. The size of \( K_g \) can overload the computer memory which is one of the reasons one might employ domain decomposition methods. In the FETI case, the mesh is decomposed into four smaller submeshes (see Figure 3-a) to

<table>
<thead>
<tr>
<th>Solver run time w. preproc.</th>
<th>Number of compute nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTFETI - small clusters</td>
<td>100</td>
</tr>
<tr>
<td>HTFETI Large clusters</td>
<td>120</td>
</tr>
<tr>
<td>TFETI - large domains</td>
<td>140</td>
</tr>
<tr>
<td>TFETI - small domains</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>80</td>
</tr>
</tbody>
</table>

Figure 1: The two approaches how to use HTFETI method and their comparison with TFETI for problems of size up to 3 billion DOF.

The HTFETI method is a variant of hybrid FETI methods introduced by Klawonn and Rheinbach (see, e.g., [9]) for FETI and FETI-DP. In the original approach, the authors gather a number of subdomains in clusters which provides a three-level domain decomposition approach. Each cluster consists of a certain number of subdomains and for these, a FETI-DP system is set up. The clusters are then solved by a traditional FETI approach using projections to treat the non-trivial kernels. In contrast, in HTFETI, a FETI approach is used for the subdomains in each cluster and the FETI approach using projections for clusters. In their recent publication [8], Klawon et al. show a state-of-the-art large scale simulations solving up to 42 billion of unknowns using 66% (524,288 Power BQC CPU cores) of the Argonne National Laboratory Mira machine. Our simulation results are comparable in size (30 billion of DOF), but running at smaller scale (98,304 Intel Xeon Ivy Bridge CPU cores of the EPCC Archer Cray XC30 machine). There are two main reasons, why we can solve larger problems per core: (1) hybrid parallelization with multiple subdomains per MPI process allows us to fully utilize available memory; and (2) the Archer supercomputer has more memory per CPU core (Archer - 2.6 GB per CPU core; Mira - 1 GB per CPU core).

The main advantage of HTFETI is the ability to solve problems decomposed into a very large number of subdomains. We have successfully tested our implementation in the ESPRESO library with over 3,000,000 subdomains organized into more than 4,000 clusters. This means two things: (i) extremely large problems can be solved (over 30 billion DOF); (ii) large and moderate problems (around 1 billion DOF) can be decomposed into very small subdomains which improve both memory and computational efficiency over TFETI.

To solve extremely large problems, the configuration of the HTFETI has to be different (Approach A: 1 cluster per compute node with large number of subdomains - up to several thousand) than for solving moderate problems (Approach B: 1 cluster per CPU core with small number of subdomains - up to a few hundred). In particular, Approach B can use subdomains of approximately 1,000 DOF to solve a 1 billion DOF problem on approximately 200 compute nodes with 64 GB of RAM. Solving the same problem with TFETI requires subdomains of approximately 90,000 DOF.

![Figure 2: Cantilever beam, FEM discretization.](image-url)
a) Domain Decomposition

\[
\lambda = \begin{pmatrix}
\lambda_1^{d,1} & \lambda_1^{p,1} \\
\lambda_2^{d,1} & \lambda_2^{p,1} \\
\lambda_3^{d,1} & \lambda_3^{p,1} \\
\lambda_4^{d,1} & \lambda_4^{p,1}
\end{pmatrix}
\]

b) Rigid body motions

c) Hybrid FETI - splitting of Lagrang. multipliers

Figure 3: Domain decomposition.

Avoid assembling the global objects \( K_0, f_0 \). Decomposition generally causes an increase in the number of nodes in cuts between subdomains, thus the global size of the unknowns of the decomposed problem is always bigger than the original one. In fact, it is not a problem, because any object in a FETI algorithm stored at one computational node does not have such dimension. Equation for \( i \)-th subdomain is

\[
K_iu_i = f_i - B_i^T \lambda, \quad i \in \{1, 2, 3, 4\},
\]

where \( K_i \) and \( f_i \) with the same meaning as before are assembled for all subdomains separately. On the RHS beside the vector \( f_i \) the product \( B_i^T \lambda \) appears. It is actually an additive nodal force vector which acts on the interface between subdomains and it substitutes influence transmitting from neighbors. Those four systems in (2) are supplemented by the global constraint condition

\[
\sum_{i=1}^{4} B_iu_i = 0.
\]

Equilibrium equations together with condition (3) can be written globally as

\[
\begin{pmatrix}
K_1 & O & O & O \\
O & K_2 & O & B_2^T \\
O & O & K_3 & B_3^T \\
B_1 & B_2 & B_3 & B_4 & O
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4
\end{pmatrix}
= \begin{pmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4
\end{pmatrix}
\]

or shortly

\[
Ku + B^T \lambda = f,
\]

\[
Bu = c.
\]

The meaning of symbols in (5) is obvious compared to detailed expression in (4). Generally, vector \( c \) contains zero entries. Vector of LM

\[
\lambda = \left( (\lambda_1^{d,1})^T, (\lambda_2^{d,1})^T, (\lambda_3^{d,1})^T, (\lambda_4^{d,1})^T \right)^T
\]

consists of four parts where the first subvector \( \lambda_1^{d,1} \) enforces Dirichlet boundary condition, the second one \( \lambda_2^{d,12} \) enforces connectivity between subdomains 1 and 2, etc. The division of global \( \lambda \) into four parts does not relate to the number of subdomains. These four sets correspond to four cuts caused by the decomposition. The splitting into four subvectors shows how \( \lambda \) can be stored according to numeric superindexes. For instance, the first part \( \lambda_1^{d,1} \) is related to the first subdomain only, and it will never touch another one, therefore it is stored only on the first subdomain. The second part \( \lambda_2^{d,12} \) connects subdomain 1 and 2, thus they are stored only there. Evidently, there is no need to assemble the global \( \lambda \) vector on a single node.

In the next step, the primal variables will be eliminated. From Eq. (5) the vector of unknown displacements is

\[
u = K \ast (f - B^T \lambda) + R \alpha,
\]

where \( R = \text{diag}(R_1, R_2, R_3, R_4) \) is block-diagonal matrix, the columns of which define the basis of null space of \( K \), and the vector \( \alpha \) contains its amplitudes. Mutual relation between them is

\[
KR = O \quad \text{or} \quad R^T K = O.
\]

The term (6) with the second condition in (5) eliminates the primal variables

\[
BK^T f - BK^T B^T \lambda + BR \alpha = c
\]

similarly, as a combination of the first equation from (5) and Eq. (7)

\[
R^T Ku + R^T B^T \lambda = R^T f.
\]

The last two terms together can be expressed as

\[
F \alpha + G^T \alpha = d,
\]

\[
G \lambda = e
\]

in which the following substitutions are used

\[
F = BK^T, \quad G = -BR^T,
\]

\[
d = BK^T f, \quad e = -R^T f.
\]

### 2.2 Projected Conjugate Gradient Method

In the previous steps, the primal variables \( u \) were eliminated. The newly derived system of linear equations (10) can be favorably solved by iterative methods, e.g., the conjugate gradient method (CGM).

1. set: \( \varepsilon > 0 \), \( i_{\text{max}} > 0 \), \( \lambda_0 \),
2. \( g_0 = g_0 + G^T \alpha_0 \) where \( g_0 = F \lambda_0 - d \)
3. \( w_0 = g_0 \)
4. do \( i = 0, 1, ..., i_{\text{max}} \)
5. \( \rho_i = -(g_0, e)/ (w_i, Fw_i) \)
6. \( \lambda_{i+1} = \lambda_{i+1} + w_i \rho_i \)
7. \( g_{i+1} = g_{i+1} + G^T \alpha_i \), \( g_{i+1} = g_{i+1} + Fw_i \rho_i \)
8. if \( ||g_{i+1}|| < \varepsilon \) break
9. end do
10. \( \gamma_i = (g_{i+1}, e)/(g_i, e) \)
11. \( w_i = D_i + w_i \gamma_i \)
12. Algorithm 1.
An approximation of LM used in Algorithm 1 in \( i \)-th iteration is considered in the form
\[
\lambda_i = \lambda_0 + \sum_{j=0}^{i} w_j \rho_j, \tag{13}
\]
If an initial guess is chosen as a linear combination of basis vectors of matrix \( G^T \) in the form of \( \lambda_0 = G^T y \), the vector \( y \) can be simply determined using the second equation in (10) as
\[
\lambda_0 = (GG^T)^{-1} e. \tag{14}
\]
Such initial guess fully accomplishes the second equality in (10), therefore the rest of the approximation in (13) lies in the kernel of \( G \).

During the iterative process, just the partial gradient \( g_i \) is kept in the memory, and before using the complete form \( G_{3:11} \) it is modified by \( \alpha_i \) to satisfy condition \( G_{3:11} \alpha_i = 0 \). Firstly, it appears if the initial conjugate vector \( w_0 \) is used as a contribution to the \( \lambda_0 \). To keep \( w_0 \) in the kernel of \( G \), vector
\[
\alpha_0 = -(GG^T)^{-1} G (F \lambda_0 - d) \tag{15}
\]
has to be evaluated to satisfy the condition
\[
G w_0 = GG_0 = G (F \lambda_0 - d) + GG^T \alpha_0 = 0. \tag{16}
\]
The fulfilment of the condition above is equivalent to the projection of the part of the gradient
\[
w_0 = g_0 = P(F \lambda_0 - d) = P g_0 \tag{17}
\]
by the orthogonal projector
\[
P = I - G^T (GG^T)^{-1} G. \tag{18}
\]
Due to the projection applied to each iteration, Algorithm 1 is called Projected Conjugate Gradient Method (PCGM) (for more details see [6]). The iterative process can be accelerated by Lumped or Dirichlet preconditioner, in this case the preconditioned gradient has to be projected again (for more detail see, e.g., [10]).

### 2.3 From FETI to HTFETI

FETI algorithm described in subsection 2.1 is efficient (besides other effects related to the condition of the system etc.) until CP
\[
GG^T \in \mathbb{R}^{n_{cp} \times n_{cp}} \tag{19}
\]
can be effectively factorized from the time point of view. Its dimension \( n_{cp} \) in linear elasticity depends on the number of subdomains \( n_{sub} \) (in 2D: \( n_{cp} = 3 \cdot n_{sub} \), in 3D: \( n_{cp} = 6 \cdot n_{sub} \)), and it is equal to all RBM in a decomposed structure. In Figure 3, as illustrated using the FETI technique, each subdomain of the decomposed beam has three RBM (in 2D), therefore \( n_{cp} = 3 \cdot 4 = 12 \). CP dimension reduction by HTFETI can be demonstrated even in this small and simple benchmark. The permutation and splitting of matrix \( B \) according to Figure 3-c reads
\[
\begin{pmatrix}
K_1 & O & O & O & B_{0,1}^T & O & B_{1,3}^T \\
O & K_2 & O & O & B_{0,2}^T & O & B_{1,2}^T \\
O & O & K_3 & O & O & B_{0,3}^T & B_{1,3}^T \\
O & O & O & K_4 & O & B_{0,4}^T & B_{1,4}^T \\
B_{0,1} & B_{0,2} & O & O & O & O & O \\
O & O & B_{0,3} & B_{0,4} & O & O & O \\
B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} & O & O & O \\
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
\lambda_0 \\
\lambda_0 \\
\lambda_1 \\
\end{pmatrix} =
\begin{pmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
c_{0,1} \\
c_{0,2} \\
c_1 \\
\end{pmatrix}
\]
where matrix
\[
B_0 = \begin{pmatrix}
B_{0,1} & B_{0,2} & O & O \\
O & O & B_{0,3} & B_{0,4} \\
\end{pmatrix}
\]
consists of constraints gluing subdomains into 2 clusters, and
\[
B_1 = \begin{pmatrix}
B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} \\
\end{pmatrix}
\]
contains the rest of the equality constraints. This permuted system of the linear equation still has the structure for a FETI algorithm as in (4). The next equation shows the system after the permutation
\[
\begin{pmatrix}
K_1 & B_{0,1}^T & O & O & O & B_{1,1}^T \\
O & K_2 & B_{0,2}^T & O & O & B_{1,2}^T \\
O & O & K_3 & O & B_{0,3}^T & B_{1,3}^T \\
O & O & O & K_4 & O & B_{0,4}^T & B_{1,4}^T \\
B_{0,1} & B_{0,2} & O & O & O & O & O \\
O & O & B_{0,3} & B_{0,4} & O & O & O \\
B_{1,1} & B_{1,2} & O & B_{1,3} & B_{1,4} & O & O \\
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
\lambda_0 \\
\lambda_0 \\
\lambda_1 \\
\end{pmatrix} =
\begin{pmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
c_{0,1} \\
c_{0,2} \\
c_1 \\
\end{pmatrix}
\]
This system can be rewritten in a simplified form
\[
\begin{pmatrix}
\tilde{K}_1 & \tilde{B}_{1,1}^T \\
O & K_2 & B_{2,1}^T \\
B_1 & B_2 & O \\
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
\lambda \\
\end{pmatrix} =
\begin{pmatrix}
f_1 \\
f_2 \\
c_1 \\
\end{pmatrix}
\]
which can be solved by the HTFETI method. The classical FETI method belongs to the group of dual Schur complement methods in which the primal variables \( u \) are eliminated and satisfied exactly in each iteration. Dual variables \( \lambda \), reaction forces between subdomains, are obtained by an iterative solver. In the case of HTFETI, not only the primal variables but also a subset of dual variables \( \lambda_0 \) are eliminated. This modification causes subdomains 1 and 2 to be partially glued onto interface into cluster 1, and subdomains 3 and 4 into cluster 2. After that, the structure behaves like a problem decomposed into two parts, and the dimension of CP is two times smaller accordingly \( n_{cp} = 3 \cdot n_c = 3 \cdot 2 = 6 \) where \( n_c \) is the number of clusters. The important thing is that the linear system (21) can be handled by the same methodology applied onto (4) in subsections 2.1 and 2.2, so long as each object is relabeled by a tilde.

### 2.4 HTFETI - Optimization

In the following text the methodology will be described on the first cluster and its subdomains 1, 2.

To keep optimal properties of the HTFETI code, the cluster stiffness matrix \( \tilde{K}_1 \) can not be assembled, but used implicitly. It requires modified routines for factorization and kernel detection.

The factorization is obtained by solving \( \tilde{K}_1 \tilde{x} = \tilde{b} \) more detailed written as
\[
\begin{pmatrix}
K_{1,2} & B_{0,1,2}^T \\
B_{0,1,2} & O \\
\end{pmatrix}
\begin{pmatrix}
x \\
\mu \\
\end{pmatrix} =
\begin{pmatrix}
b \\\nz \\
\end{pmatrix}
\]
with \( K_{1,2} = \text{diag}(K_1, K_2) \) and \( B_{0,1,2} = (B_{0,1}, B_{0,2}) \). Notation 1 : 2 points to the first and last ordinal number of the subdomains on the cluster. So for the second cluster it would be 3 : 4. The system (22) can be taken as a FETI problem to be solved not iteratively but by a direct solver.
After the dualization it reads

\[
\begin{pmatrix}
F_{0,1:2}^T & G_{0,1:2}^T \\
G_{0,1:2} & 0
\end{pmatrix}
\begin{pmatrix}
\mu \\
\beta
\end{pmatrix}
= 
\begin{pmatrix}
d_{0,1:2} \\
e_{0,1:2}
\end{pmatrix}
\tag{23}
\]

with substitutions

\[
F_{0,1:2} = B_{0,1:2}^T \cdot K_{+}^T \cdot B_{0,1:2}^T - z, \quad G_{0,1:2} = -R_{1,2}^T \cdot B_{0,1:2}^T, \\
d_{0,1:2} = B_{0,1:2} \cdot b, \quad e_{0,1:2} = -R_{1,2}^T \cdot b.
\]

To obtain the vector \( \tilde{x} = (x^T, \mu^T)^T \), both systems (22), (23) are subsequently solved in three steps

\[
\begin{align*}
\beta &= S_{0,1:2}^+ (G_{0,1:2} F_{0,1:2}^{-1} g_{0,1:2} - e_{0,1:2}) \\
\mu &= F_{0,1:2}^{-1} (g_{0,1:2} - G_{0,1:2}^T \beta) \\
x &= K_{1:2} (b - B_{0,1:2} \mu) + R_{1:2} \beta
\end{align*}
\tag{24}
\]

in which singular Schur complement

\[
S_{0,1:2} = G_{0,1:2} F_{0,1:2}^{-1} G_{0,1:2}^T
\]

appears.

In the HTFETI method, not only do the subdomain matrices \( K_i \) have to be factorized, but also \( F_{0,1:2} \) and \( S_{0,1:2} \) - one pair on each cluster. The dimension of \( F_{0,1:2} \) is controlled by the number of LM which glue subdomains of the cluster, and the dimension of \( S_{0,1:2} \) is multiplied by the number of subdomains per cluster multiplied by the defect of \( K_i \).

To get the kernel \( R_1 \) of the first cluster, the following term is written

\[
\begin{pmatrix}
K_1 & O \\
O & K_2
\end{pmatrix}
\begin{pmatrix}
B_{0,1}^T & B_{0,2}^T \\
B_{0,1}^T & B_{0,2}^T
\end{pmatrix}
\begin{pmatrix}
R_1 & O \\
O & R_2
\end{pmatrix}
\begin{pmatrix}
H_1 & O \\
O & H_2
\end{pmatrix} = 
\begin{pmatrix}
O & O \\
O & O
\end{pmatrix}
\]
\tag{25}

or shortly

\[
\begin{pmatrix}
K_{1,2} & B_{0,1,2}^T \\
B_{0,1} & O
\end{pmatrix}
\begin{pmatrix}
R_{1,2} & O \\
O & O
\end{pmatrix}
H_{1:2} = 
\begin{pmatrix}
O & O \\
O & O
\end{pmatrix}
\]
\tag{26}

where the kernel is given by

\[
R_1 = (R_{1:2}^T, O)^T H_{1:2}.
\]

Assuming that the subdomain kernels \( R_1 \) and \( R_2 \) are already known, the determination of \( H_{1:2} \) remains. The first equation

\[
K_{1:2} R_{1:2} H_{1:2} = O
\]

from (26) does not impose any special conditions onto \( H_{1:2} \), because the product \( K_{1:2} R_{1:2} \) is already a zero matrix. The second equation reads

\[
B_{0,1:2} R_{1:2} H_{1:2} = -G_{0,1:2}^T H_{1:2} = O
\]
\tag{27}

and it says, \( H_{1:2} \) is kernel of \( G_{0,1:2}^T \). It can be beneficial if kernels of subdomains stored in \( R_{1:2} \) are obtained analytically and from one coordinate system (see, e.g., eq. (3.7) in [2]). Then the searched kernel is

\[
H_{1:2} = (I_{3,3}, I_{3,3})^T
\]
\tag{28}

where \( I_{3,3} \in \mathbb{R}^{3\times3} \) is an identity matrix. Matrix \( H_{1:2} \) is also the kernel of \( S_{0,1:2} \), which can be regularized by it and then easily factorized.

3. Parallel Implementation

Both the TFETI and HTFETI algorithms have been implemented in our ESPRESO library [1]. ESPRESO uses hybrid parallelization in the form of a combination of MPI and Cilk++. In this section we describe, the main challenges that we have encountered when developing a solver for extremely large problems.

Due to Cholesky decomposition of the stiffness matrices, all FETI algorithms are more efficient when using subdomains with small number of DOF. The main reasons are: (i) reduced memory utilization associated with storing the Cholesky decomposition data, (ii) reduced processing time of forward and backward substitution (please note that in this text this operation is also called the solve routine of the SpDS). From the numerical point of view, the subdomains with smaller number of DOF provide better conditioning overall, therefore the iterative solver requires a lower number of iterations to find a solution.

However, solving large problems using subdomains with small number of DOF generates a large number of subdomains and a very large coarse problem. For TFETI, the coarse problem size is defined by the number of subdomains. The ESPRESO library is able to use the TFETI method with approximately 50,000 subdomains solving 4 billion DOF. This can be achieved with the use of our distributed approaches (i) to assemble the CP and (ii) to calculate its inverse matrix. For more details see Section 3.2.1. In the case of the HTFETI, the CP size is given by the number of clusters. In ESPRESO one cluster is processed by one MPI process. Therefore, the CP size is given by the number of MPI processes. If MPI process is executed per node, the CP size is given by the number of compute nodes. Since the same technique is used with TFETI, the solver should be able to run on a supercomputer of up to 50,000 nodes.

The results presented in this paper show that the HTFETI implementation in ESPRESO can efficiently run on up to 4 096 clusters/nodes (tested on EPCC Archer) or 4913 clusters/nodes (tested on CSCS Piz Daint) using clusters with 1000 subdomains.

3.1 Shared Memory Parallelization

The main goal of shared memory parallelization is to find optimal configuration of the cluster in terms of: (i) the number of subdomains per cluster, (ii) the subdomain size, and (iii) the number of clusters per compute node. This section also describes the memory efficient implementation of the Lumped preconditioner.

3.1.1 Properties of the HTFETI Objects

The approach used for the shared memory parallelization is based on assigning multiple subdomains to single MPI process. The number of subdomains has to be equal to or larger than the number of CPU cores per node divided by the number of MPI processes per node. A cluster cannot be processed by more than one compute node. However, more clusters can be processed by a single node, because MPI standard allows multiple processes to be executed per node.

Each cluster contains objects that are specific for each subdomain: \( K_i, R_i, B_{0,i}, B_{1:i}, e_i, T_i, P_i \). There are also objects related to the HTFETI algorithm that are common to all subdomains per cluster: \( F_0, S_0, G_0 \). In addition, several objects that belong to subdomains are combined and stored per cluster: \( G \) and \( GG^T \) matrices (these two matrices
Table 1: Processing time reduction achieved by using $K^+_i \cdot B_{0,i}$ object in HTFETI method for cluster of size 5.5 millions DOF.

<table>
<thead>
<tr>
<th>Num. of subdomains [-]</th>
<th>64</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subdomain size [DOF]</td>
<td>30^2</td>
<td>15^2</td>
<td>12^2</td>
</tr>
<tr>
<td>$K^+_i$ SpDS solve time [s]</td>
<td>0.951</td>
<td>0.448</td>
<td>0.366</td>
</tr>
<tr>
<td></td>
<td>97.2%</td>
<td>92.0%</td>
<td>85.5%</td>
</tr>
<tr>
<td>$K^+<em>i \cdot B</em>{0,i}^T$ GEMV time [s]</td>
<td>0.025</td>
<td>0.026</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td>2.6%</td>
<td>5.4%</td>
<td>6.5%</td>
</tr>
<tr>
<td>$F_0$, $S_0$ processing time [s]</td>
<td>0.002</td>
<td>0.012</td>
<td>0.035</td>
</tr>
<tr>
<td></td>
<td>0.2%</td>
<td>2.6%</td>
<td>8.0%</td>
</tr>
</tbody>
</table>

are global objects and they are described in Section 3.2).

There are two main parameters that affect the processing time and memory utilization. We have to evaluate their effect on both preprocessing and per iteration time to find the optimal configuration. These parameters are:

- subdomain size - affects the size of stiffness matrix $K_i$, and most importantly the size of the Cholesky or LU decomposition.
- number of subdomains per cluster - affects the size of a cluster coarse problem $S_0$ and $F_0$.

The following notes describe the main properties of the most important HTFETI objects that have to be taken into consideration:

Note 1: $F_0$ is a sparse matrix and has to be factorized with a SpDS during the preprocessing stage. During the iterative solver runtime, the solve routine of the SpDS with one RHS is called twice per iteration.

Note 2: $S_0$ is a dense matrix and it is factorized by the dense LAPACK routines during the preprocessing stage. During the iterative solver runtime, the solve routine is called once per iteration;

Note 3: $F_0$ and $S_0$ are objects that break the parallelization over multiple subdomains. In every iteration the following sequence $y = F_0^{-1} x$; $z = S_0^{-1} y$ and $x = F_0^{-1} z$ is executed in between two parallel regions. Since all three operations execute Gaussian elimination with single RHS, these are not parallelized. The amount of processing time spent in this sequence is shown in Table 1. The table shows that it is becoming a notable bottleneck by means of Amdahl's law for a high number of subdomains per cluster.

Note 4: During the preprocessing stage, when object $F_0$ is assembled, a set of Schur complements in the form of $S_{ci} = B_{0,i} \cdot K_i^+ \cdot B_{0,i}^T$ is calculated. These $S_{ci}$ matrices are placed into $S_0$ to a particular position to form the final matrix. By keeping the dense matrices ($K_i^+ \cdot B_{0,i}^T$) stored in the memory as separate objects, the single iteration time can be reduced significantly. In the action of the HTFETI $K_i^+$ operator, the SpDS solve routine on $K_i^+$ object is called twice. Please note that this is the most time consuming operation in every iteration. One of these two operations can be replaced by the dense matrix-vector multiplication with the $(K_i^+ \cdot B_{0,i})$ matrix. The amount of time saved is shown in Table 1.

3.1.2 Optimal Cluster Configuration

All tests were performed on Cray XC40 compute node of the CSC Sisu supercomputer with two 12-core Intel (Xeon) Haswell (E5-2690v3) processors running at 2.6 GHz clock rate and 64 GB of RAM.

Optimal memory utilization. First, we will evaluate the cluster configurations in terms of memory utilization. The cluster sizes have been selected so that 1, 2, 4, 8, and 12 clusters can fit to a single compute node’s memory. This means that the total memory footprint of a single cluster (or an MPI process) has to be smaller than 64, 32, 16, 8, and 5.3 GB of RAM. The goal is to find a configuration for solving the largest possible problem per cluster.

Please note that if the number of subdomains is small, then the number of DOF per subdomain has to be large to maintain the required cluster size. This yields low memory efficiency caused by the Cholesky decomposition of $K_i$ matrices. On the other hand, the large number of subdomains yields large $F_0$ and $S_0$ matrices, which has to be factorized as well.

The Figure 4 shows the results of this test. Several observations can be made out of this figure. First, the clusters with smaller number of subdomains are more memory efficient - this means that if more smaller clusters are executed per one node, a bigger problem per node can be solved. This, on the other hand, increases the size of the TFETI CP. Second, large clusters require higher number of subdomains to reach best memory efficiency. This is due to the fact that only a large number of subdomains yields reasonable subdomain sizes. The optimal subdomain size is between 3000 to 6500 DOF. In particular, the following subdomain sizes are optimal [cluster size in millions DOF / subdomain size in DOF]: 0.68/3000; 1.1/3000; 2.2/5194; 4.3/6591, and 7.1/6591.

Optimal Solver Performance. Second, we will evaluate the HTFETI performance in terms of (i) preprocessing time, (ii) single iteration time, and (iii) iterative solver runtime which includes both parallel and numerical scalability.

The single iteration describes the optimal configuration to minimize the execution time of the $K^+$ operator action in the HTFETI ignoring numerical behavior. This is shown in Figure 5 where optimal solutions are highlighted by larger markers. Figure 4 and 5 show that the configuration that minimizes the execution time is also the optimal one in terms of memory utilization in 4 out of 5 cases. In the remaining case, the 7.1 million cluster size, the variation in performance is less than 4%.
The solver runtime includes both the number of iteration and the single iteration time. This is shown in Figure 5. It can be observed that the trend for all cluster configurations is very similar to the single iteration time. This is due to the fact that the number of iterations in all configurations close to the optimal one differs only between 2% to 10%.

The HTFETI preprocessing time evaluation is shown in Figure 6. This figure shows that processing time to calculate the HTFETI objects $F_0$ and $S_0$, in general, increases with the number of subdomains per cluster. The higher processing time for a very small number of subdomains is caused by the calculation of the $F_0$ object which needs to solve very large systems $K^i_0$ with $B_0^i$ matrix as the RHS. As this is the only one-time paid penalty during the preprocessing, the optimal configuration for HTFETI preprocessing is usually not used in favor of optimal configuration for a single iteration.

3.1.3 Memory Efficient Lumped Preconditioner

The lumped preconditioner is a sparse matrix object which is equal to the stiffness matrix without regularization. Therefore, there is no additional cost associated with its assembling. Since a SpDS requires regularized stiffness matrix available all the time, it has to be stored in the memory in this form.

Please note that the $K_0$ matrices are the second largest objects stored per subdomain (the largest object is Cholesky decomposition of the $K_0$). Therefore, it is important not to have two copies of this object stored in the memory. Since the $K_0 = K_0 + T_0$, it is more efficient to store the regularized $K_0$ and $T_0$. The $T_0$ is a sparse matrix with a very small number of non-zero elements, negligible when compared to number of non-zeros in $K_0$. Please see the memory requirements of both objects in Table 2.

The action of the lumped preconditioner is then transformed from a single SpMV $x_{out} = K_0 \cdot x_{in}$ to two SpMV calls in the form of $x_{out} = K_0 \cdot x_{in} - T_0 \cdot x_{in}$. The performance overhead of this method is between 6.7% to 12.5% as shown in Table 2.

This feature is implemented in the ESPRESSO library to allow users to process larger problems per single node, if required. If maximum performance is a priority, ESPRESSO can keep a copy of non-regularized stiffness matrix in the memory.

3.2 Cluster Parallelization

3.2.1 Coarse Problem Preprocessing and Projector Implementation

The action of FETI coarse problem $GG^T$ in the projector $P$ is a global operation in FETI. However, its assembling process can be performed in a nearest neighbor fashion. This approach is essential in order to be able to assemble the $GG^T$ matrix in parallel if number of subdomains is in order of tens of thousands. First the approach for TFETI is described, followed by specifics for HTFETI.

The basic case is the TFETI with single subdomain per MPI process. Suppose each subdomain/MPI process has its own matrix $G_i$. In addition, it has to obtain the matrices $G_i$ from all of its neighboring subdomains. Then the subdomains multiply the local $G_i$ matrix with all neighboring $G_i$ matrices one by one to obtain submatrices of the $GG^T$. Suppose the local subdomain has index $i$ and neighboring subdomain has index $j$. Then the submatrix is calculated as $GG^T(i,j) = G_i G_j^T$. The size of this submatrix is equal to $d \times d$, where $d$ is the defect of matrix $K_i$ (for 3 dimensional case of linear elasticity $d = 6$). The location where
The generator is using hybrid parallelization in form of MPI and Cilk++. Matrices $K$, $M$ for FEM and BEM discretization.

Based on the mesh data, the assembler generates essential objects for TFETI and HTFETI solvers, such as: stiffness matrix $K_i$, right-hand side $f_i$, gluing matrix $B_i$, matrix of HTFETI corners $B_0$, a set of fix nodes required for regularization of matrix $K_i$ for TFETI method. Matrix $R_i$, the subdomain kernel, is generated by the (H)TFETI solver itself. In the current version, the assembler contains packages for FEM and BEM discretization.

The generator is using hybrid parallelization in form of MPI and Cilk++. Matrices $K_i$, $M$ and RHS $f$ are assembled independently for each sub-domain using multiple threads. The assembler of the gluing matrix $B_i$ combines final matrix out of 3 blocks: (i) the Dirichlet boundary conditions, (ii) the remaining MPI processes. Each MPI process then performs a Cholesky decomposition of the $G G^T$ and calculates only one small part of the $(G G^T)^{-1}$ that is locally required by the projector $P$.

This approach efficiently parallelizes the most critical part of the calculation of the inverse matrix: the forward and backward substitution for every column of the identity matrix $I$ the dimension of which is equal to dimension of $G G^T$. The minor penalty of this method is that all MPI processes have to perform Cholesky decomposition. However, this operation is significantly less expensive than several thousands executions of the forward and backward substitution.

3.3 Parallel Problem Generator

The ESPRESO library contains a parallel problem generator and parallel matrix assembler that generate data for the solver. The generator provides a way of fast evaluation of the solver behavior for extremely large linear elasticity problems in the form of a cube of size 30 mm, as shown in Figure 10. The cube is made from a steel with following parameters: Young’s modulus $E = 2.1 \cdot 10^5$ MPa, Poisson’s ratio $\mu = 0.3$, density $\rho = 7850$ kg/m$^3$, gravity constant $g_{x1} = 9.81$ m/s. The cube is fixed on the plane $x_1 = 0$, and loaded by its own weight in $x_1$ direction. The discretization is done using 8 node brick elements.

The process starts with generation of cubical meshes for each cluster. In the second step, the cluster meshes are decomposed into subdomains using either METIS or geometric decomposition into cubical subdomains.

Based on the mesh data, the assembler generates essential objects for TFETI and HTFETI solvers, such as: stiffness matrix $K_i$, right-hand side $f_i$, gluing matrix $B_i$, matrix of HTFETI corners $B_0$, a set of fix nodes required for regularization of matrix $K_i$ for TFETI method. Matrix $R_i$, the subdomain kernel, is generated by the (H)TFETI solver itself. In the current version, the assembler contains packages for FEM and BEM discretization.

The generator is using hybrid parallelization in form of MPI and Cilk++. Matrices $K_i$, $M$ and RHS $f$ are assembled independently for each sub-domain using multiple threads. The assembler of the gluing matrix $B_i$ combines final matrix out of 3 blocks: (i) the Dirichlet boundary conditions, (ii)
4. RESULTS

In order to evaluate the potential of the HTFETI solver on large scale problems, we are using the parallel generator described in Section 3.3. The decomposition is geometrical so that the cubic problem is further decomposed into the cubic cluster and these are decomposed into cubic subdomains.

4.1 Weak Scalability

The first test evaluates the weak scalability. It is performed on EPCC Cray XC 30 Archer supercomputer with total of 4920 compute nodes. These are connected using Cray Aries interconnect in a Dragonfly topology. The node configuration is as follows: two 2.7 GHz, 12-core E5-2697v2 (Ivy Bridge) series processors; 64 GB of memory shared between the two processors.

The HTFETI solver configuration is: a single cluster per compute node; 729 subdomains per cluster; 12288 DOF per node and approximately 7.5 millions of DOF per cluster/compute node. These settings allowed us the best memory utilization for this particular node configuration. The stopping criteria for the HTFETI solver is set 0.0001 in dual variables and the lumped preconditioner is applied.

Figure shows the overall processing time divided into four main parts: (i) $K$ regularization and factorization; (ii) HTFETI preprocessing; (iii) FETI preprocessing and (iv) conjugate gradient solver runtime. The (i) (ii) and (iv) belongs to preprocessing stage of the linear solver. Parts (i) and (ii) are embarrassingly parallel problems and their processing time remains constant for the entire weak scalability evaluation. The HTFETI preprocessing is the penalty for using the HTFETI method as it assembles all the extra HTFETI operations

<table>
<thead>
<tr>
<th>Problem size $[10^9]$ DOF</th>
<th>3.8</th>
<th>7.4</th>
<th>9.8</th>
<th>12.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel efficiency [-]</td>
<td>100%</td>
<td>97%</td>
<td>97%</td>
<td>95%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem size $[10^9]$ DOF</th>
<th>16.2</th>
<th>20.3</th>
<th>24.2</th>
<th>30.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel efficiency [-]</td>
<td>95%</td>
<td>94%</td>
<td>94%</td>
<td>94%</td>
</tr>
</tbody>
</table>
The advantages of HTFETI in this configuration are: short time of $K$ factorization due to very small subdomains; (ii) FETI preprocessing time (CP preprocessing) comparable to TFETI with very large subdomains; (iii) single iteration time is significantly shorter than (2.8 to 5.1 times shorter than TFETI with small subdomains; 4.9 times shorter than TFETI with large subdomains).

The first disadvantage of HTFETI method is the worse convergence, i.e. higher number of iterations required to find a solution. The number of iterations for all three cases is: (i) TFETI with small subdomains 22 iterations; (ii) TFETI with large subdomains 33 iterations; and HTFETI 60 iterations (iii). However, the short single iteration time eliminates this bottleneck and its CG solver is the most efficient. The second disadvantage of the HTFETI method is the HTFETI preprocessing time. This remains constant for the weak scalability evaluation.

4.2 Strong Scalability

The strong scalability tests have been performed on the CSCS Piz Daint Cray XC30 supercomputer with a total of 5,272 compute nodes. The compute nodes are equipped with one 8-core Intel SandyBridge CPU (Intel Xeon E5-2670) with 32 GB of memory. For this test a problem of 2.6 billion DOF is solved from 1,000 to 4,913 compute nodes. To reduce the processing time, the stopping criteria was set to $10^{-3}$. This explains the lower number of iterations and shorter overall solver runtime when compared to the weak scalability test where the stopping criteria was $10^{-4}$. The cluster configuration for this test was: 1 cluster per node; 64 subdomains per cluster; the subdomain size from 46,875 DOF for 1,000 nodes to 6,591 DOF for 4,913 nodes. The results are presented in Figure 13 which shows that the HTFETI solver has the potential to achieve the linear strong scaling for up to 5,000 compute nodes.

5. CONCLUSION

This paper shows the potential of the HTFETI method and its implementation in our ESPRESO library. As this paper is mainly focused on the capabilities of the HTFETI solver itself, our parallel problem generator is used to generate very large problems. This generator has been used for both weak and strong scalability evaluations using the 3D linear elasticity cube benchmark.

The weak scalability efficiency of the solver is shown in Table 3. This table takes into account the entire solver runtime including the preprocessing stage. The detailed values for this test are shown in Figure 11. This test proves that HTFETI can solve problems of size over 30 billion of DOF and shows that larger problems can be solved if larger machine is available.

On the other hand, the strong scalability evaluation shows the linear scalability of the solver from 1,000 to 4,913 compute nodes. This was achieved mainly due to our parallel implementation of the coarse problem processing using the explicit inverse matrix.

We have also shown that HTFETI has great potential for "smaller problems" of up to 3 billion of DOF, providing significantly shorter time to solution and better scalability, as illustrated in Figure 1.

6. FUTURE WORK

We are currently working to support the Intel Xeon Phi and GPGPU accelerators using the dense representation of the stiffness matrices in the form of Schur complement. We are also developing the support for real world problems generated from commercial and open-source engineering tools.

7. ACKNOWLEDGMENT

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 LM 2015070".

---

**Figure 12:** The weak scalability of the HTFETI solver measured on CSC Sisu supercomputer for smaller problems of size from 5.3 millions to 1.7 billion DOF (1 to 343 compute nodes).

**Figure 13:** The strong scalability evaluation of the HTFETI solver measured on the CSCS Piz Daint supercomputer. Problem of size 2.6 billion DOF is solved on 1000 to 4,913 compute nodes. The single iteration time (top) and solver runtime (bottom).
8. REFERENCES


