Hybrid Parallelization of the Total FETI Solver

Lubomír Říha[1], Tomáš Brzobohatý[1], and Alexandros Markopoulos[1]

[1]IT4Innovations, VSB Technical University of Ostrava, Czech Republic

Abstract

This paper describes our new hybrid parallelization of the Finite Element Tearing and Interconnecting (FETI) method for the multi-socket and multi-core computer cluster. This is an essential step in our development of the Hybrid FETI solver were small number of neighboring subdomains is aggregated into clusters and each cluster is processed by a single compute node.

In our previous work we have implemented FETI solver using MPI parallelization into our ESPRESO solver. The proposed hybrid implementation provides better utilization of resources of modern HPC machines using advanced shared memory runtime systems such as Cilk++ runtime. Cilk++ is an alternative to OpenMP which is used by ESPRESO for shared memory parallelization.

We have compared the performance of the hybrid parallelization to MPI-only parallelization. The results show that we have reduced both solver runtime and memory utilization. This allows a solver to use a larger number of smaller sub-domains and in order to solve larger problems using a limited number of compute nodes. This feature is essential for users with smaller computer clusters.

In addition, we have evaluated this approach with large-scale benchmarks of size up to 1.3 billions of unknowns to show that the hybrid parallelization also reduces runtime of the FETI solver for these types of problems.

Keywords: ESPRESO, Total FETI, Hybrid Parallelization, MPI, Cilk++.

1 Introduction

The goal of this paper is to describe the Hybrid parallelization of FETI method based on our variant of the Finite Element Tearing and Interconnecting (FETI) type domain decomposition method called Total FETI (TFETI) [6]. The original FETI method, also called the FETI-1 method, was originally introduced for the numerical solution of large linear systems arising in linearized engineering problems by Farhat and

Roux [1]. In the FETI methods, a body is decomposed into several non-overlapping subdomains and the continuity between the subdomains is enforced by Lagrange multipliers. Using the theory of duality, a smaller and relatively well conditioned dual problem can be derived and efficiently solved by a suitable variant of the conjugate gradient algorithm.

The original FETI algorithm, where only the favorable distribution of the spectrum of the dual Schur complement matrix [10] was considered, was efficient only for a small number of subdomains. So it was later extended by introducing a natural coarse problem [12, 13], whose solution was implemented by auxiliary projectors so that the resulting algorithm became, in a sense, optimal [12, 13]. Even if there are several efficient coarse problem parallelization strategies [14], still there are size limitations of the coarse problem. Thus several hybrid (multilevel) methods were proposed [15, 16].

The key idea is to aggregate a small number of neighboring subdomains into the clusters, which naturally results in the smaller coarse problem. In our Hybrid Total FETI [17], the aggregation of subdomains into the clusters is enforced again by Lagrange multipliers. Hybrid FETI method uses two level decomposition: the problem is decomposed into clusters (the first level), then the clusters are decomposed into subdomains (the second level).In the TFETI method [6], also the Dirichlet boundary conditions are enforced by Lagrange multipliers. Hence all subdomain stiffness matrices are singular with apriori known kernels which is a great advantage in the numerical solution. With known kernel basis we can effectively regularize the stiffness matrix [7] and use any standard Cholesky type decomposition method for nonsingular matrices.

This paper is an extended version of the paper [2] presented at the Fourth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering. Further evaluation of the hybrid parallelization in the ESPRESO FETI Solver for large scale problems, up to 1.3 billion of unknowns, is described in Section 4.3. In this section we also identify under what conditions the main bottleneck of the FETI method, the coarse problem, becomes unacceptable. We also present our current effort, which deals with reducing the coarse problem size using the Hybrid FETI method in Section 5.

1.1 Total FETI method

The FETI-1 method [6] is based on the decomposition of the spatial domain into non-overlapping subdomains that are glued by Lagrange multipliers, enforcing arising equality constraints by special projectors. The original FETI-1 method assumes that the boundary subdomains inherit the Dirichlet conditions from the original problem. This means, that subdomains touching Dirichlet boundary are fixed while others are kept floating; in linear algebra speech, corresponding subdomain stiffness matrices are non-singular and singular, respectively. The basic idea of our Total-FETI (TFETI) [6] is to keep all the subdomains floating and enforce the Dirichlet boundary conditions by means of a constraint matrix and Lagrange multipliers, similarly to the gluing conditions along subdomain interfaces. This simplifies the implementation of the stiffness matrix pseudoinverse. The key point is that kernels of subdomain stiffness matrices are known a priori, have the same dimension and can be formed without any computation from mesh data. Furhermore, each local stiffness matrix can be regularized cheaply, and the inverse of the resulting nonsingular matrix is at the same time a pseudoinverse of the original singular one [7–9].

Let N_p, N_d, N_n, N_c denote the primal dimension, the dual dimension, the null space dimension and the number of cores available for our computation. Primal dimension means the number of all DOF including those arising from duplication on the interfaces. Dual dimension is the total number of all equality and inequality constraints. Let us consider a partitioning of the global domain Ω into N_S subdomains $\Omega^s, s = 1, \ldots, N_S$ $(N_S \geq N_c)$. Subdomain stiffness matrix $\mathbf{K}^s$ and the subdomain nodal load vector $\mathbf{f}^s$ correspond to each subdomain Ω^s. $\mathbf{R}^s$ shall be a matrix whose the columns of which span the nullspace (kernel) of $\mathbf{K}^s$. Let $\mathbf{B}^s$ be a signed boolean matrix defining connectivity of the subdomain s with neighbour subdomains. It also enforces Dirichlet boundary conditions when TFETI is used. They constitute global objects

$$\begin{aligned}
\mathbf{K} &= \text{diag}(\mathbf{K}^1, \ldots, \mathbf{K}^{N_S}) \in \mathbb{R}^{N_p \times N_p},\\
\mathbf{R} &= \text{diag}(\mathbf{R}^1, \ldots, \mathbf{R}^{N_S}) \in \mathbb{R}^{N_p \times N_n},\\
\mathbf{B} &= [\mathbf{B}^1, \ldots, \mathbf{B}^{N_S}] \in \mathbb{R}^{N_d \times N_p},\\
\mathbf{f} &= [(\mathbf{f}^1)^T, \ldots, (\mathbf{f}^{N_S})^T]^T \in \mathbb{R}^{N_p \times 1}.
\end{aligned} \tag{1}$$

Note that columns of $\mathbf{R}$ also span the kernel of $\mathbf{K}$.

Let us apply the duality theory to the primal problem

$$\min \frac{1}{2}\mathbf{u}^T\mathbf{K}\mathbf{u} - \mathbf{u}^T\mathbf{f} \quad \text{s.t.} \quad \mathbf{B}\mathbf{u} = \mathbf{o} \tag{2}$$

and let us establish the following notation

$$\mathbf{F} = \mathbf{B}\mathbf{K}^\dagger\mathbf{B}^T, \quad \mathbf{G} = -\mathbf{R}^T\mathbf{B}^T, \quad \mathbf{d} = \mathbf{B}\mathbf{K}^\dagger\mathbf{f}, \quad \mathbf{e} = -\mathbf{R}^T\mathbf{f},$$

where $\mathbf{K}^\dagger$ denotes a pseudoinverse of $\mathbf{K}$, satisfying $\mathbf{K}\mathbf{K}^\dagger\mathbf{K} = \mathbf{K}$, and $\mathbf{G}$ is a so-called *natural coarse space matrix*. We obtain a new QP

$$\min \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{F}\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\mathbf{d} \quad \text{s.t.} \quad \mathbf{G}\boldsymbol{\lambda} = \mathbf{e}. \tag{3}$$

Furthermore, equality constraints $\mathbf{G}\boldsymbol{\lambda} = \mathbf{e}$ can be homogenized to $\mathbf{G}\boldsymbol{\mu} = \mathbf{o}$ by splitting $\boldsymbol{\lambda}$ into $\boldsymbol{\mu} + \widetilde{\boldsymbol{\lambda}}$ where $\widetilde{\boldsymbol{\lambda}}$ satisfies $\mathbf{G}\widetilde{\boldsymbol{\lambda}} = \mathbf{e}$. This implies $\boldsymbol{\mu} \in \operatorname{Ker}\mathbf{G}$. The vector $\widetilde{\boldsymbol{\lambda}}$ can be chosen as the least square solution of the equality constraints, i.e. $\widetilde{\boldsymbol{\lambda}} = \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{e}$. We substitute $\boldsymbol{\lambda} = \boldsymbol{\mu} + \widetilde{\boldsymbol{\lambda}}$, minimize over $\boldsymbol{\mu}$ (terms without $\boldsymbol{\mu}$ can be omitted) and add $\widetilde{\boldsymbol{\lambda}}$ to $\boldsymbol{\mu}$.

Finally, equality constraints $\mathbf{G}\boldsymbol{\mu} = \mathbf{o}$ can be enforced by the orthogonal projector $\mathbf{P} = \mathbf{I} - \mathbf{Q}$ onto the null space of $\mathbf{G}$, where $\mathbf{Q} = \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{G}$ is the orthogonal projector onto the image space of $\mathbf{G}^T$ (i.e. $\operatorname{Im}\mathbf{Q} = \operatorname{Im}\mathbf{G}^T$ and $\operatorname{Im}\mathbf{P} = \operatorname{Ker}\mathbf{G}$). The final problem reads

$$\mathbf{PF}\boldsymbol{\mu} = \mathbf{P}\widetilde{\mathbf{d}}, \tag{4}$$

where $\widetilde{\mathbf{d}} = \mathbf{d} - \mathbf{F}\widetilde{\boldsymbol{\lambda}}$. Note we call the action of $(\mathbf{GG}^T)^{-1}$ the *coarse problem* of FETI.

Lemma 1 *The matrix* $\mathbf{PF}$ *is symmetric positive definite on* $\mathrm{Ker}\mathbf{G}$.

Reader can find the proof of Lemma 1 in [12]. Thanks to the Lemma 1 the problem (4) may be solved efficiently by the PCG (Preconditioned Conjugate Gradients) algorithm. The conjugate gradient method is a good choice thanks to the classical estimate by Farhat, Mandel, and Roux [12] of the spectral condition number:

$$\kappa(\mathbf{PFP}|\mathrm{Im}\mathbf{P}) \leq C\frac{H}{h}. \tag{5}$$

2 Preconditioners

This section briefly summarizes FETI preconditioners introduced in [4]. The extension for cases with discretization heterogenities can be found in [5]. The subdomain's stiffness matrix $\mathbf{K}^{(s)}$ can be divided into four blocks

$$\mathbf{K}^{(s)} = \begin{bmatrix} \mathbf{K}_{ii}^{(s)} & \mathbf{K}_{ib}^{(s)} \\ \mathbf{K}_{bi}^{(s)} & \mathbf{K}_{bb}^{(s)} \end{bmatrix}, \tag{6}$$

according to indexes i (internal unknowns) and b (boundary unknowns). The Dirichlet preconditioner reads

$$\mathbf{F}^{D^{-1}} = \sum_{s=1}^{N_s} \mathbf{B}^{(s)} \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{S}_{bb}^{(s)} \end{bmatrix} \left(\mathbf{B}^{(s)}\right)^T, \tag{7}$$

where $\mathbf{S}_{bb}^{(s)}$ is the Schur complement of the block $\mathbf{K}_{ii}^{(s)}$ with respect to the subdomain stiffness matrix $\mathbf{K}^{(s)}$,

$$\mathbf{S}_{bb}^{(s)} = \mathbf{K}_{bb}^{(s)} - \left(\mathbf{K}_{ib}^{(s)}\right)^T \left(\mathbf{K}_{ii}^{(s)}\right)^{-1} \mathbf{K}_{ib}^{(s)}. \tag{8}$$

The Dirichlet preconditioner is numerically scalable. The second 'lumped' preconditioner lumps the Dirichlet operator on the substucture interface unknowns

$$\mathbf{F}^{L^{-1}} = \sum_{s=1}^{N_s} \mathbf{B}^{(s)} \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{K}_{bb}^{(s)} \end{bmatrix} \left(\mathbf{B}^{(s)}\right)^T. \tag{9}$$

The preconditioner $\mathbf{F}^{L^{-1}}$ is not numerically scalable, but it is more economical than the Dirichlet preconditioner. The matrix $\mathbf{B}^{(s)}$ may not have a full column rank as redundant constrains are allowed. To enable the use of the Dirichlet or lumped preconditioner, a scaling matrix $\mathbf{A}^{(s)}$ has to be introduced. The matrix $\mathbf{A}^{(s)}$ is a diagonal square matrix, its size equals the number of dual variables, and its diagonal entries are given by the weight vector $\mathbf{w}$. After the decomposition, each node on the cut is split into at least two nodes. In such case, $\mathbf{w}(i) = 1/2$ for each of its associated degrees of freedom i. Generally, if the number of subdomains associated with a node is m, the corresponding entry of the weight vector is $\mathbf{w}(i) = 1/m$. The modified forms of the Dirichlet and lumped preconditioners are

$$\begin{aligned} \mathbf{F}^{D^{-1}} &= \sum_{s=1}^{N_s} \mathbf{A}^{(s)}\mathbf{B}^{(s)} \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{S}_{bb}^{(s)} \end{bmatrix} \left(\mathbf{B}^{(s)}\right)^T \mathbf{A}^{(s)} \\ \mathbf{F}^{L^{-1}} &= \sum_{s=1}^{N_s} \mathbf{A}^{(s)}\mathbf{B}^{(s)} \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{K}_{bb}^{(s)} \end{bmatrix} \left(\mathbf{B}^{(s)}\right)^T \mathbf{A}^{(s)}. \end{aligned} \tag{10}$$

Obviously, these preconditioners have almost the same computational complexity as the basic ones (7), (9). The modified conjugate gradient algorithm with the preconditioner and projector (PCGP) reads as follows. By the symboll $\overline{\mathbf{F}^{-1}}$ we denote any FETI preconditioner. In our case, it is actually substituted by $\mathbf{F}^{L^{-1}}$ or $\mathbf{F}^{D^{-1}}$ from (10).

Algorithm 1 (Linear solver based on the TFETI method)

```
1:  Set $\mathbf{G} := -\mathbf{R}^T\mathbf{B}^T$, $\mathbf{H} := (\mathbf{G}\mathbf{G}^T)^{-1}$, $\mathbf{d} := \mathbf{B}\mathbf{K}^\dagger\mathbf{f}$, and $\mathbf{e} := -\mathbf{R}^T\mathbf{f}$.
2:  Compute $\tilde{\boldsymbol{\lambda}} := \mathbf{G}^T\mathbf{H}\mathbf{e}$.
3:  Set $\tilde{\mathbf{d}} := \mathbf{d} - \mathbf{F}\tilde{\boldsymbol{\lambda}}$.
4:  Compute $\boldsymbol{\mu}$ from (4) by PCGP:
5:      $\mathbf{r}^0 = \tilde{\mathbf{d}},\ \boldsymbol{\mu}^0 = \mathbf{o}$.
6:      for $j = 1, 2, \ldots,$ until convergence do
7:          Project $\mathbf{w}^{j-1} = \mathbf{P}_\mathbf{G}\mathbf{r}^{j-1}$.
8:          Precondition $\mathbf{z}^{j-1} = \overline{\mathbf{F}^{-1}}\mathbf{w}^{j-1}$.
9:          Re-project $\mathbf{y}^{j-1} = \mathbf{P}_\mathbf{G}\mathbf{z}^{j-1}$.
10:         if $j = 1$
11:             $\beta^j = 0,\ \mathbf{p}^j = \mathbf{y}^0$.
12:         else
13:             $\beta^j = (\mathbf{y}^{j-1})^T\mathbf{w}^{j-1}/(\mathbf{y}^{j-2})^T\mathbf{w}^{j-2}$.
14:             $\mathbf{p}^j = \mathbf{y}^{j-1} + \beta^j\mathbf{p}^{j-1}$.
15:         end
16:         $\gamma^j = (\mathbf{y}^{j-1})^T\mathbf{w}^{j-1}/(\mathbf{p}^j)^T\mathbf{F}\mathbf{p}^j$.
17:         $\boldsymbol{\mu}^j = \boldsymbol{\mu}^{j-1} + \gamma^j\mathbf{p}^j$.
18:         $\mathbf{r}^j = \mathbf{r}^{j-1} - \gamma^j\mathbf{F}\mathbf{p}^j$.
19:         if $\|\mathbf{w}^{j-1}\| \le \epsilon_{PCGP}\|\mathbf{r}^0\|$ then stop.
20:     end for
```

```
21:     $\boldsymbol{\mu} = \boldsymbol{\mu}^j$.
22: Set $\boldsymbol{\lambda} := \tilde{\boldsymbol{\lambda}} + \boldsymbol{\mu}$.
23: Compute $\boldsymbol{\alpha} := \mathbf{HG}(\mathbf{d} - \mathbf{F}\boldsymbol{\lambda})$.
24: Compute $\mathbf{u} := \mathbf{K}^{\dagger}(\mathbf{f} - \mathbf{B}^T\boldsymbol{\lambda}) + \mathbf{R}\boldsymbol{\alpha}$.
```

3 ESPRESO Solver with Hybrid Parallelization

The ESPRESO Total FETI solver is implemented in C++. A significant part of the development effort was devoted to writing a C++ wrapper for (1) the selected sparse and dense BLAS routines and (2) the sparse direct solvers (MKL version of PARDISO direct solver) of the Intel MKL library. By a simple modification of this wrapper, we can add support for additional direct solvers as needed.

Since the solver development is mainly focused on the current and future multi and many core architectures, in particular the Intel MIC architecture, the Intel MKL library is the only external tool used by the solver. In addition, to be able to port the solver to Intel Xeon Phi (in both native and offload mode) the Intel compiler is used for compilation and Intel MPI is used as message passing library.

The hybrid parallelization inside the ESPRESO solver is designed to fit the two-level decomposition used by the Hybrid FETI Method. This method decomposes the problem into clusters (the first level), then the clusters are decomposed into subdomains (the second level). In the case of the Total FETI method the problem is decomposed into subdomains only, but subdomains are processed in groups. In this paper we focus on the Total FETI method only.

In our implementation this decomposition is mapped to parallel hardware in a following way. Clusters (for HFETI) or groups of subdomains (for TFETI) are mapped to compute nodes of a supercomputer, therefore a parallelization model for distributed memory is used - in our case the message passing (MPI). Subdomains inside a clusters or groups are mapped to CPU cores of the particular compute node, therefore a shared memory model is used for the second level.

Current implementation allows to process multiple clusters/groups by a single compute node, but single cluster/group cannot be processed by more than one node, as this is a general limitation of the shared memory parallelization.

There are two major parts of the solver that affects its parallel performance and scalability: (1) communication layer (described in Section 3.1) and (2) the inter-node processing routines for shared memory (described in Section 3.2).

The first part deals with the optimization of the communication overhead caused mainly by gluing matrix $\mathbf{B}$ multiplication and application of the projector (includes multiplication with matrix $\mathbf{G}$ and the application of the coarse problem). Having a fully optimized communication layer is essential for scalability. The second part is a set of routines developed for efficient parallel processing of multiple subdomains in shared memory.

3.1 MPI Communication Layer

The hybrid parallelization is well suited for multi-socket and multi-core compute nodes. This is a hardware configuration used by most of today's supercomputers.

The first level of parallelization is designed for parallel processing of the groups of sub-domains. Each group is assigned to a single node but if necessary multiple groups can be processed per node. As mentioned earlier multiple nodes cannot work on a single group. The distributed memory parallelization is done using MPI. In particular, we are using MPI standard 3.0 (implemented in the Intel MPI 5.0) because the communication hiding techniques implemented in our FETI communication layer require the non-blocking collective operations.

The essential part of this parallelization is the development of efficient communication layer. This layer is identical, whether the Total FETI solver runs single or multiple domains per group. It uses novel communication avoiding and hiding techniques for the main iterative solver. In particular, we have implemented: (1) the Pipelined Conjuagent Gradient (PipeCG) iterative solver - hides communication cost of the global dot products in CG behind the local matrix vector multiplications; (2) the coarse problem solver using distributed inverse matrix - merges two global communication operations (Gather and Scatter) into one (AllGather) and parallelizes the coarse problem processing; and (3) the optimized version of global gluing matrix multiplication (matrix $\mathbf{B}$ for FETI) - implemented as a stencil communication which is fully scalable.

The stencil communication for a simple decomposition into four sub-domains is shown in Figure 1 where the Lagrange Multipliers (LMs) that connect different neighboring subdomains are depicted in different colors. In every iteration when LMs are updated an exchange is performed between the neighboring sub-domains to finish the update. This affinity also controls the distribution of the data for the main distributed iterative solver, which in our case iterates over local LMs only. In our implementation each MPI process modifies only those elements of the vectors that match the LMs associated with all domains in its respective group.

We call this operation the vector compression. In the pre-processing stage the local part of the gluing matrix $\mathbf{B}$ is also compressed using the same approach (in this case it means that all the empty rows are removed from the matrix) so that we can directly use sparse matrix vector multiplication on the compressed vectors.

3.2 Inter-node Processing

The second level of parallelization is designed for parallel processing of sub-domains in a group. Our implementation enables oversubscription of CPU cores in a way that each core can process multiple sub-domains. This means that the number of subdomains processed per compute node is not limited by its hardware configuration.

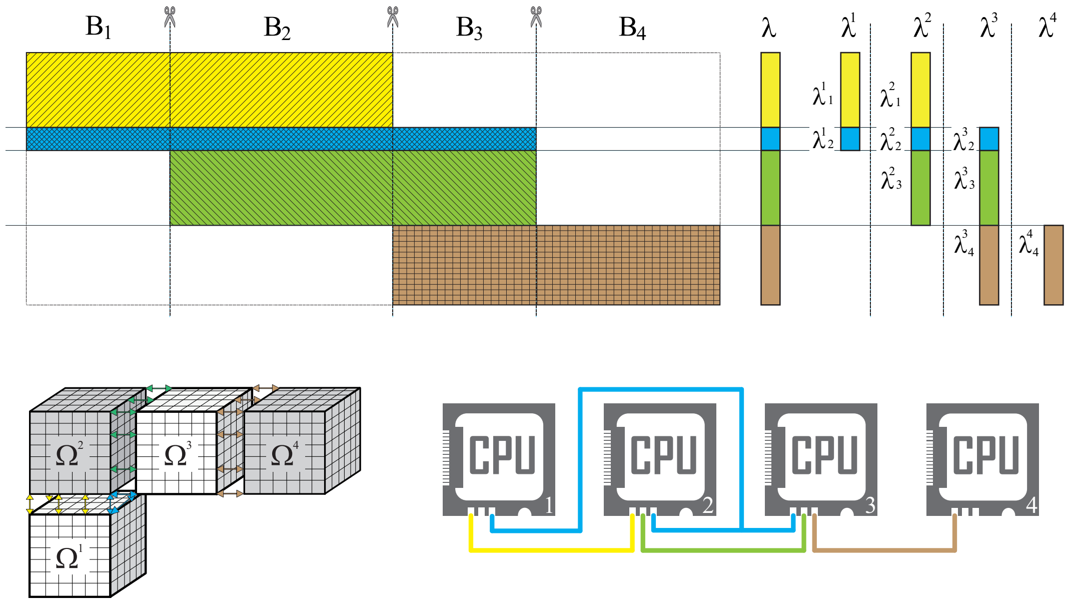


Figure 1: Stencil communication in FETI defined by the gluing matrix $\mathbf{B}$

If the number of sub-domains per group is less than the number of cores, then multiple MPI processes per node must be executed in order to utilize all the CPU cores.

The shared memory parallelization is implemented using Intel Cilk++. We have chosen the Cilk++ due to its advanced support for C++ language. In particular, we are taking advantage of the functionality that allows us to create custom parallel reduction operations on top of the C++ objects, which in our case, are sparse matrices.

4 Numerical experiments

We have evaluated the implementation on the solution of the following 3D linear elasticity problems: (1) Synthetic 3D-cube benchmark and (2) Real-world engine benchmark (both described in the following sections). The performance evaluation was carried out on Anselm supercomputer located at IT4Innovations in the Czech Republic. The machine has the following parameters:

- IT4Innovation's Anselm
  - up to 3,300 cores
  - non-blocking cluster of 209 nodes each with:
    * 2x 8-core Intel Sandy Bridge E5-2665 (Sandy Bridge), 2.4GHz and 64GB of RAM

∗ InfiniBand QDR network - 40 Gbit/s inter-node bandwidth

4.1 Real world benchmark - Evaluation of the communication layer

The first benchmark is a 2.5 million DOF model of a car engine depicted in Figure 2. Using this benchmark we have evaluated the behavior of the communication layer during a strong scaling test. We have run the benchmark decomposed into 32, 64, 128, 256, 512, and 1024 subdomains in the TFETI mode only on Anselm supercomputer.

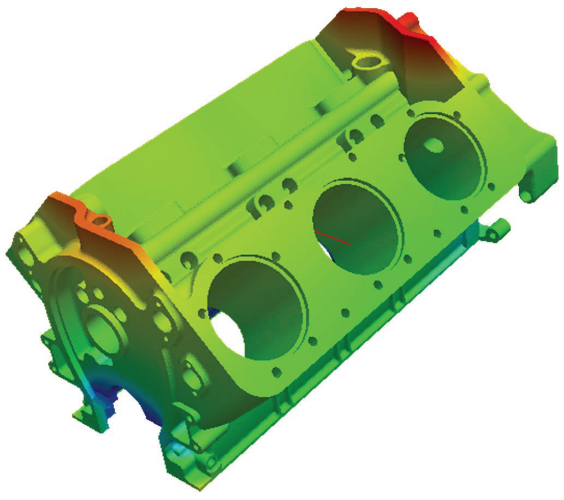


Figure 2: The car engine benchmark

Figure 3 shows how the two optimization techniques implemented in the communication layer and the application of the simple lumped preconditioner help the scalability and solver performance in terms of single iteration time. Note that all these tests ran with 8 MPI processes per node. As we do not have a preconditioned pipelined CG algorithm implemented yet, we have evaluated two most efficient combinations: (1) regular CG with the lumped preconditioner and (2) pipelined CG without preconditioner, and the effect of using a distributed inverse matrix of the coarse problem. As expected, preconditioned regular CG has slower iterations starting from 32 subdomains but more importantly starting from 256 subdomains GGTINV starts making the difference. This effect becomes dominant for decomposition into 512 subdomains and essential in the case of 1024 subdomains as preconditioned regular CG with GGTINV is faster than the pipelined CG without proconditioning. Reader should note, that in both cases using GGTINV keeps the scaling superlinear up to 1024 subdomains.

The advantage of using the lumped preconditioner is shown in Figure 4 where, on average, the number of iterations is reduced by 60% - 70%. When these numbers are combined with the per iteration time, we get the entire solution time, shown in Figure 5. In this figure the significant iteration reduction achieved using the lumped

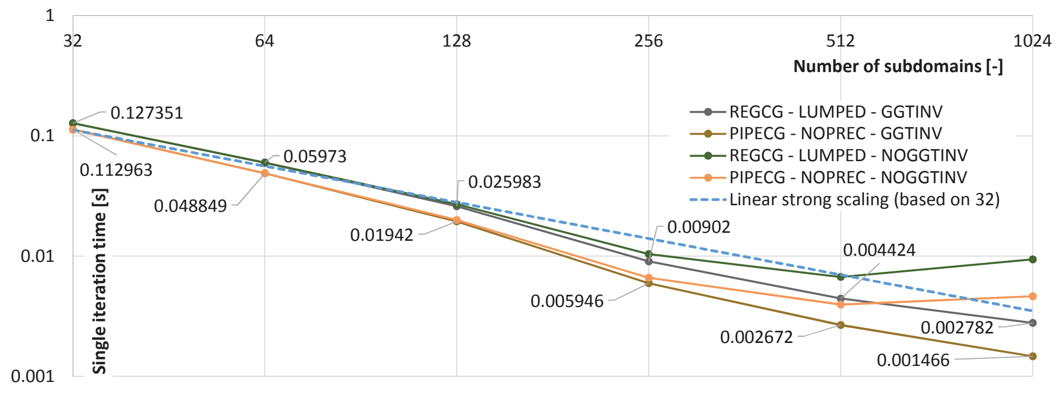


Figure 3: Strong scaling evaluation of the real problem ”Engine 2.5 Milions DOF” benchmark for the TFETI method. Single iteration time for pipelined CG (PIPECG) and regular CG (REGCG) with and without the lumped preconditioner (LUMPED and NOPREC) and a distributed inverse matrix of the coarse problem (GGTINV and NOGGTINV).

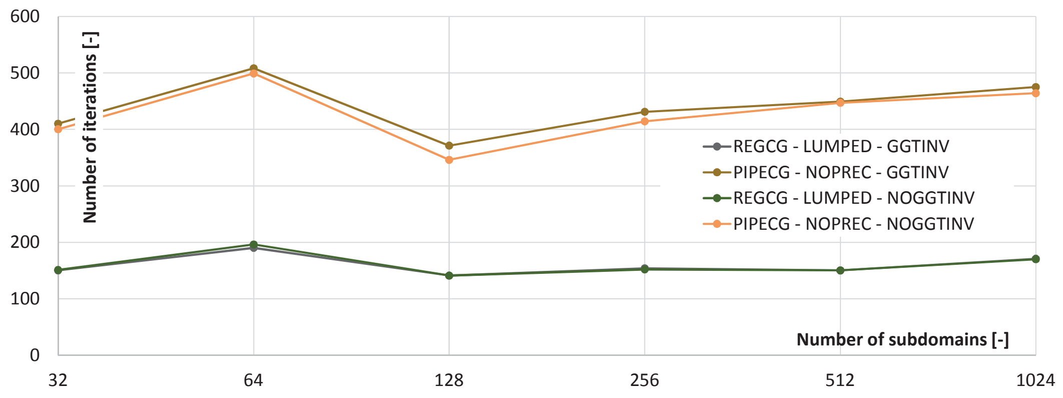


Figure 4: Strong scaling evaluation of the real problem ”Engine 2.5 Milions DOF” benchmark for the TFETI method. Number of iterations for different number of subdomains with and without the lumped preconditioner.

preconditioner is seen from very beginning but gets eliminated by iteration time for decompositions into 512 and 1024 subdomains where using GGTINV becomes the most significant aspect of the entire solution time. Again, the most important information is that we are able to achieve the superlinear scaling for the entire solution up to 1024 subdomain problem decomposition using simple preconditioner.

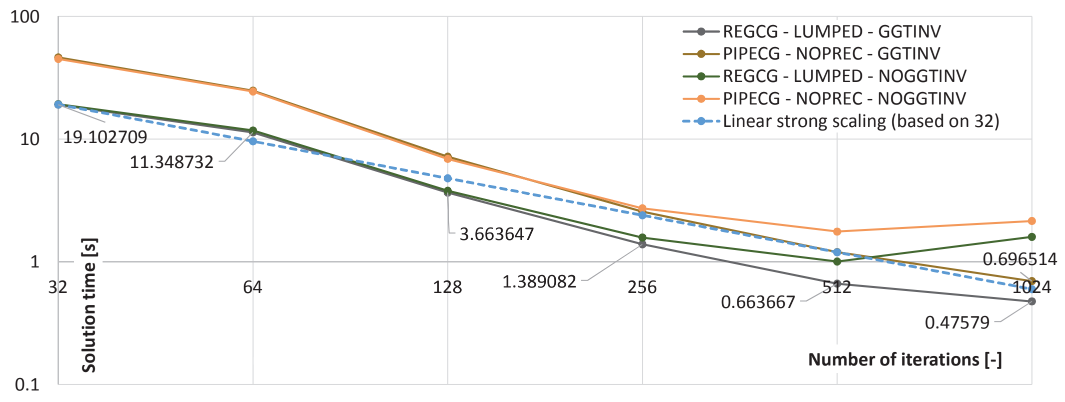


Figure 5: Strong scaling evaluation of the real problem "Engine 2.5 Milions DOF" benchmark for the TFETI method. Entire solution time in seconds for different number of subdomains.

4.2 Synthetic 3D cube benchmark - Evaluation of the inter-node processing

The second benchmark is a linear elasticity problem in a three-dimensional domain. The domain depicted in Figure 6 has a shape of a cube with the length of the edge 1m. We considered fixed steel box deformed only by its own weight. We prescribe Dirichlet boundary condition $u_x = u_y = u_z = 0$ on the left wall. All other walls are free. The material constants are defined by the Young modulus $E = 2.1 \times 10^5$ [MPa], the Poisson ratio $\nu = 0.3$.

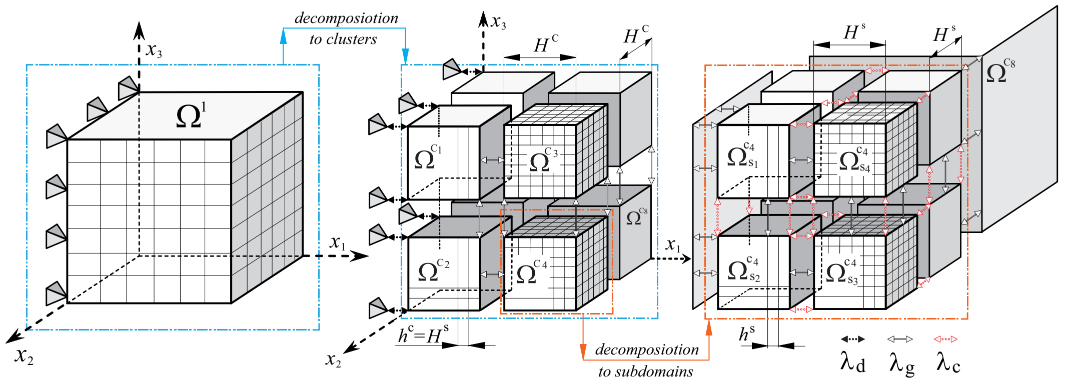


Figure 6: CUBE Benchmark - decomposition into groups of subdomains, where each group is processed by a single compute node.

This test is focused on solving the largest possible problem using limited hardware resources. The following measurements were performed on 8 nodes of the Anselm cluster, each with 94 GB of RAM. The evaluation criteria are: (1) memory usage and (2) overall processing time including preprocessing and solver runtime.

Limited memory size dictates the maximum sub-domain size, as large domains produce large L and U factors. Therefore the sub-domain sizes must remain reasonable.

Another limiting factor is the amount of MPI processes that can run on a certain number of nodes and CPU cores. 8 compute nodes, each with 16 cores, can run up to 128 MPI processes without over subscribing the CPU cores with multiple MPI processes, which is not recommended.

The solution to the limiting number of concurrently running MPI processes is the proposed hybrid parallelization. This allows the solver to run a single MPI process per node and to use Cilk++ runtime for shared memory parallelization. Cilk++ runtime will execute sub-domain processing routines in a way that all CPU cores will be utilized but will not be forced to run multiple routines at the same time, which leads to unnecessary context switching.

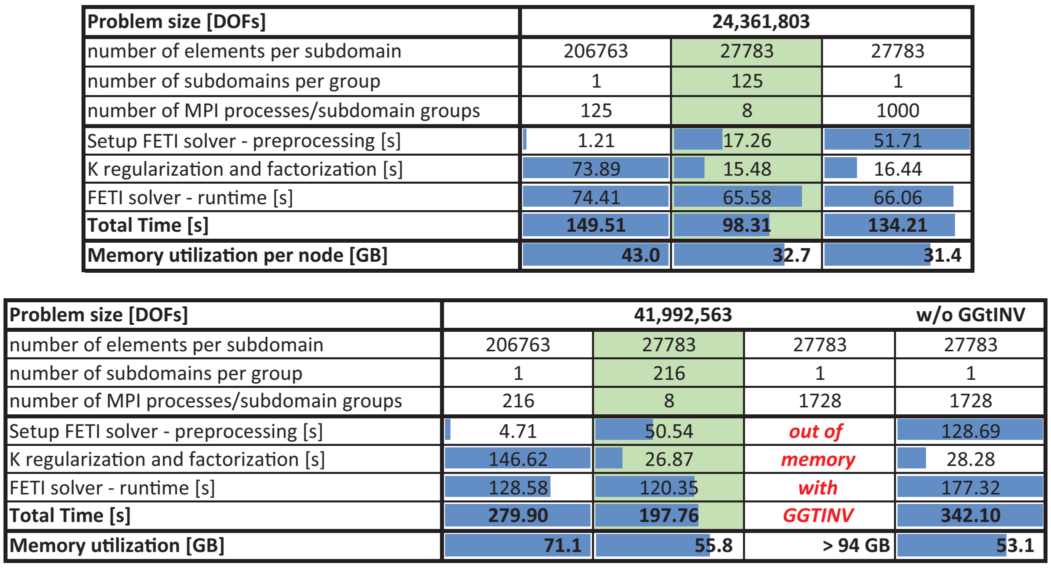
The results of these tests are shown in Figure 7. We have run the experiments for 2 problem sizes 24,361,803 DOF and 41,992,563 DOF both executed on the identical hardware (8 nodes of Anselm supercomputer). For each problem size, three scenarios are evaluated:

- MPI only parallelization with large subdomains (first column in Figure 7)
  - 1 subdomain per MPI process
  - number of MPI processes identical to total number of CPU cores
  - subdomain size as required by the problem size
- optimal hybrid parallelization (second column in Figure 7)
  - domain size less than 30,000 DOF (to have L and U factors of reasonable size)
  - 1 MPI process per node
  - number of subdomains per MPI process as required by the problem size
- MPI only parallelization with oversubsription of CPU cores by MPI processes (third column in Figure 7)
  - domain size less than 30,000 DOF
  - 1 subdomain per MPI process
  - number of MPI processes as required by the problem size

In terms of the preprocessing time, the most efficient configuration is the one with a small number of large subdomains. But the factorization time, which is significantly higher in this case, increases the overall processing time and makes this method the least efficient. In case of the two configurations with smaller subdomains, the solver runtime is similar and the main difference is in the preprocessing time. This is significantly lower in the case of hybrid parallelization and makes it the most efficient configuration in terms of the processing time.

Another key factor is the memory utilization. As expected, the configuration using large subdomains has the highest memory utilization due to the size of the factors. Both remaining configurations have similar utilization during the solver runtime, but there is a peak utilization during the construction of the coarse problem and calculation of its distributed inverse matrix (GGtINV). The coarse problem is constructed on all MPI processes and therefore, if a high number of MPI processes is running per node, the peak utilization is very high and the solver runs out of memory. This problem arose for the larger problem size and configuration with 1728 groups/MPI processes (216 MPI processes per node). Therefore, in this case, we have disabled the use of GGtINV which allows the solver to create the coarse problem only on the root MPI process. Please note that this slows down the solver performance.

This problem is eliminated by the hybrid parallelization, as it allows us to run single MPI process per node, and therefore only one coarse problem is constructed per node.



Problem size [DOFs]	**24,361,803**		
number of elements per subdomain	206763	27783	27783
number of subdomains per group	1	125	1
number of MPI processes/subdomain groups	125	8	1000
Setup FETI solver - preprocessing [s]	1.21	17.26	51.71
K regularization and factorization [s]	73.89	15.48	16.44
FETI solver - runtime [s]	74.41	65.58	66.06
Total Time [s]	**149.51**	**98.31**	**134.21**
Memory utilization per node [GB]	**43.0**	**32.7**	**31.4**

Problem size [DOFs]	**41,992,563**			**w/o GGtINV**
number of elements per subdomain	206763	27783	27783	27783
number of subdomains per group	1	216	1	1
number of MPI processes/subdomain groups	216	8	1728	1728
Setup FETI solver - preprocessing [s]	4.71	50.54	***out of***	128.69
K regularization and factorization [s]	146.62	26.87	***memory***	28.28
FETI solver - runtime [s]	128.58	120.35	***with***	177.32
Total Time [s]	**279.90**	**197.76**	***GGTINV***	**342.10**
Memory utilization [GB]	**71.1**	**55.8**	**> 94 GB**	**53.1**

Figure 7: Evaluation of the efficiency of the hybrid implementation of the FETI method.

4.3 Large-scale tests using 3D cube benchmark

In this section, an evaluation of the hybrid parallelization in the ESPRESO FETI Solver for large problems is presented. All tests are performed on the 3D cube benchmark described in Section 4.2. The following three configurations are evaluated:

- MPI only parallelization with large subdomains (Figure 8)

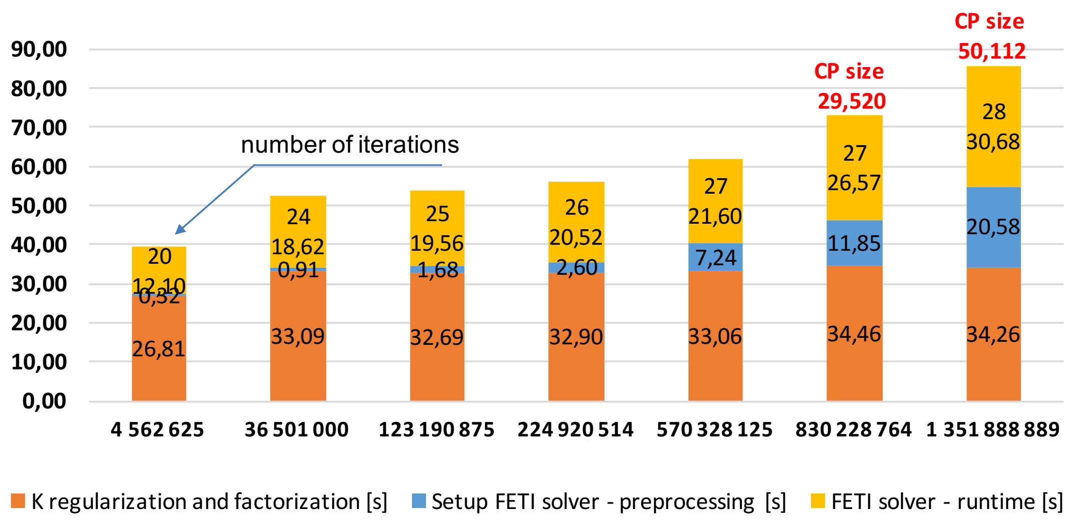


Figure 8: Solving the problem of the size of 1.3 billion unknowns using large subdomains with MPI parallelization only (1 MPI process per CPU core). (Configuration: domain size = 177,957 DOF; 1 subdomains per MPI process; 24 subdomains per node). The tests were executed on : 2, 9, 31, 56, 115, 205 and 348 nodes of SurfSara Cartesius supercomputer).

  - 1 subdomain per MPI process; 24 MPI processes per node - equal to the number of CPU cores
  - subdomain size 177,957 DOF
  - number of compute nodes: 2, 9, 31, 56, 115, 205 and 348 nodes
  - coarse problem size - up to 50,112 for 348 compute nodes

- Hybrid parallelization with medium domains (Figure 9)
  - 64 subdomains per node; 1 MPI process per node
  - domain size less than 46,875 DOF
  - number of compute nodes: 1, 8, 27, 64, 125, 216 and 343 nodes
  - coarse problem size - up to 131,172 for 348 nodes
- Hybrid parallelization with medium domains (Figure 10)
  - 729 subdomains per node; 1 MPI process per node
  - domain size less than 6,591 DOF
  - number of compute nodes: 1, 8, 27. For 64 nodes the coarse problem processing ran out of memory

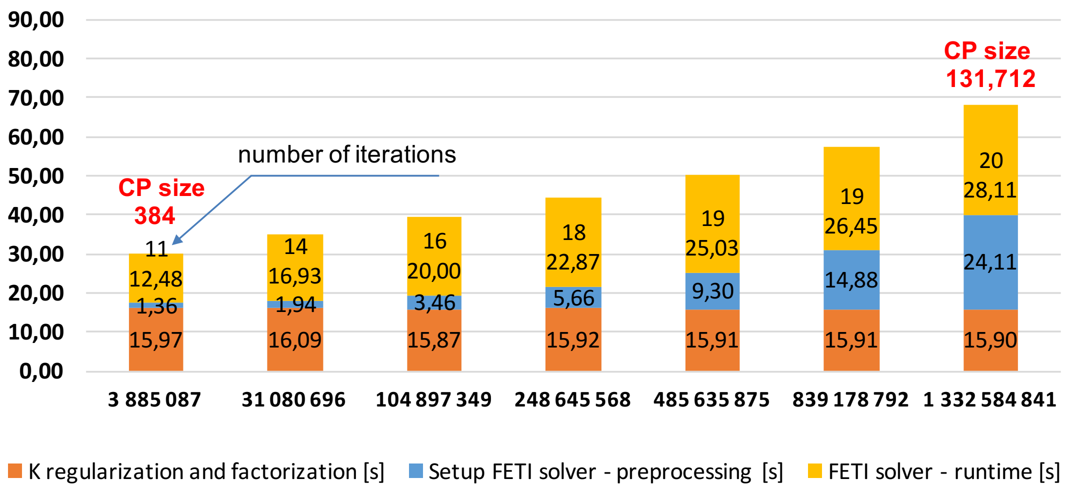


Figure 9: Solving problem of size 1.3 billion unknowns using smaller subdomains with hybrid parallelization (1 MPI process and 24 threads per node). (Configuration: domain size = 46,875 DOF; 64 subdomains per MPI process/node). Tests were executed on : 1, 8, 27, 64, 125, 216 and 343 nodes of SurfSara Cartesius supercomputer).

– coarse problem size - up to 279,936 for 54 nodes

In the case of the FETI solver, decreasing the domain size reduces the factorization time of the stiffness matrices (in the figures ”K regularization and factorization”). For the three presented cases, the time spent by factorization is reduced from 33 seconds (177,957 DOF subdomains) to 16 seconds (46,875 DOF subdomains) and ultimately to 5 seconds (6,591 DOF subdomains). Please note that in all cases the identical hardware configuration works with the problem of the identical size, only decomposition scheme (number and size of subdomains) is different.

On the other hand, increasing the number of subdomains, the coarse problem increases as well which leads to exponential growth of the coarse problem processing time (in figures this is represented with the light blue bar as ”Setup FETI solve - preprocessing”). This behaviour can be observed in all experiments (Figures 8, 9 and 10, but it is the most obvious in Figure 10. Here the 27 nodes were assembling and processing the coarse problem of the size 118,098 for 57 seconds. The large problem on 64 nodes and with coarse problem of the size 279,939 was memory demanding and therefore the solver ran out of memory. In Figure 9 the coarse problem of the size 131,712 unknowns is processed in 24 seconds due to our parallel algorithm which assembles the $\mathbf{GG}^T$ matrix in distribute fashion. But since the factorization of entire $\mathbf{GG}^T$ is done on each node, its time is not reduced.

From the overall solver runtime for 1.3 billion DOF test, it can be seen that the

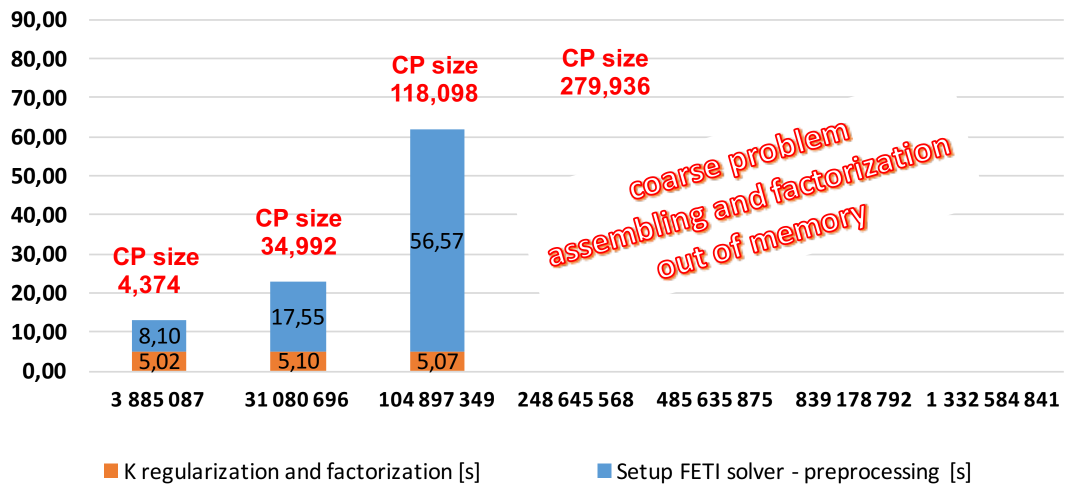


Figure 10: Preprocessing stage of the FETI solver with hybrid parallelization. The factorization time of the stiffness matrices can be further reduced by reducing the domain size and increasing the number of subdomains per node. (Configuration: domain size = 6,591 DOFs; 729 subdomains per MPI process/node). Tests were executed on : 2, 9, 27 (64) nodes of SurfSara Cartesius supercomputer).

decomposition into smaller domains is more efficient in multiple aspects. This is possible only thanks to hybrid parallelization. As the smaller domains have better condition number, the number of iterations is also smaller. In addition, the single iterations time is shorter for smaller domains, therefore the FETI solver itself is more efficient. In terms of the preprocessing, the amount of time saved by the factorization is significant. The only slower part is the coarse problem preprocessing, but only by 5 seconds. To sum up, the solver runtime is reduced from 85 to 68 seconds.

It is our intention to further reduce the preprocessing time and to avoid the main bottleneck of FETI, which is the coarse problem. The first result of this effort is shown in Figure 11. Here it can be observed that we managed to flatten the FETI preprocessing time (light blue bar). The penalty is in the form of the Hybrid FETI preprocessing. This, however, is an embarrassingly parallel operation, the processing time of which remains constant for growing number of compute nodes used for processing. A short description of this approach is in the following section.

5 Current work - Reducing the coarse problem processing time

Introduced implementation of ESPRESO parallel solver demonstrates the robustness of the FETI algorithm. Although FETI itself enables large scale problems with hundreds of millions of unknowns to be solved, it has some limitations. It can be the size of the coarse problem $\mathbf{G}\mathbf{G}^T$. In linear elasticity it is equal to all of independent rigid body modes (RBM) of all subdomains (one subdomain contributes with 6 RBM). Due to factorization or orthogonalization process its size is limited according to the computer architecture. In [17], the Hybrid Total FETI method allowing the control of the size of the coarse problem $\mathbf{G}\mathbf{G}^T$ by clusters is introduced. The main idea is based on the creation of clusters. The number of clusters can be established in advance according to the specific requirements. Then each cluster can be decomposed into smaller subdomains, which are glued together on ’corner’ nodes via special and ’small’ set of Lagrange multipliers $\boldsymbol{\lambda}_0$ calculated exactly in each iteration. Practically it means that together with primal variables $\mathbf{u}$ also the set $\boldsymbol{\lambda}_0$ is eliminated, therefore one cluster will contribute to the whole size of $\mathbf{G}\mathbf{G}^T$ only with 6 RBM, just like one subdomain in the common FETI method. The implementation of Hybrid FETI method to ESPRESO library is in progress.

6 Conclusion

The results presented in this paper show that our new hybrid parallelization of the Finite Element Tearing and Interconnecting (FETI) method for the multi-socket and multi-core computer cluster allows users having computer clusters of a limited size to solve larger problems.

When compared to MPI only parallelization the ESPRESO solver can now (1) use smaller subdomains to reduce both time and memory usage for factorization of the stiffness matrix and (2) avoid oversubscription of CPU cores with MPI processes to reduce preprocessing time.

The results measured with large benchmarks of the size up to 1.3 billions of unknowns show that hybrid parallelization also reduces runtime of the FETI solver for these types of problems. It also opens the path for developing the Hybrid FETI method that will be implemented in ESPRESO in near future. In this paper we have already shown its potential to significantly reduce the preprocessing time for the large number of subdomains.

7 Acknowledgement

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project „IT4Innovations excellence in

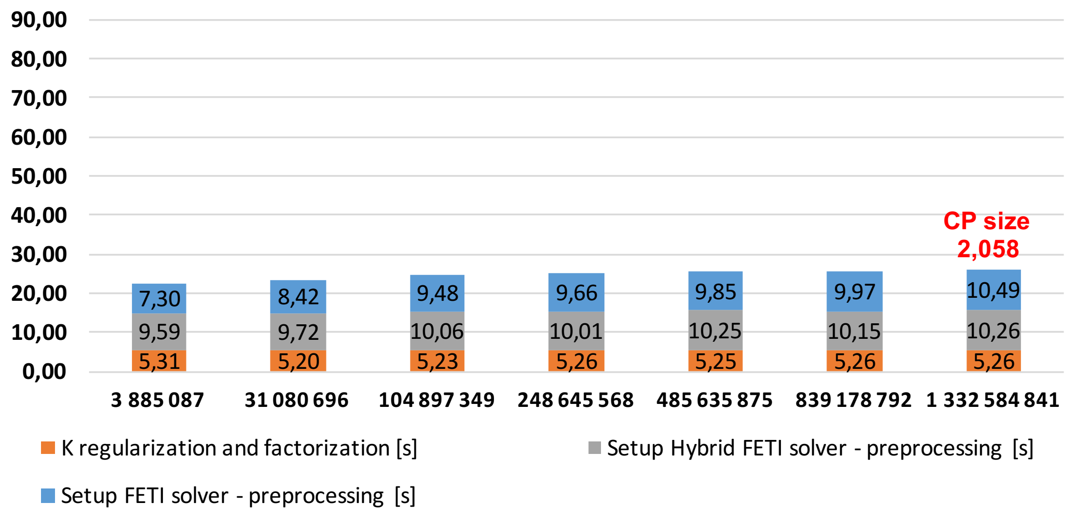


Figure 11: Significant reduction of the preprocessing time using Hybrid FETI method using MPI + Cilk parallelization. The coarse problem size is given by the number of nodes. Additional preprocessing, Hybrid FETI Setup, is required to setup the solver. This penalty, however, remains constant for any number of nodes. Problem size up to 1.3 billion unknowns. (Configuration: domain size = 6,591 DOF; 729 subdomains per MPI process/node). Tests were executed on : 1, 8, 27, 64, 125, 216 and 343 nodes of SurfSara Cartesius supercomputer).

science - LQ1602" and from the Large Infrastructures for Research, Experimental Development and Innovations project „IT4Innovations National Supercomputing Center - LM2015070", and by Grant Agency of the Czech Republic GAČR grant 13-30657P.

We thank SURFsara (www.surfsara.nl) for the support in using the Cartesius supercomputer.

References

[1] Farhat, C. and Roux, F. X.: A Method of Finite Element Tearing and Interconnecting and its Parallel Solution Algorithm. International Journal for Numerical Methods in Engineering, vol. 32, pages 1205-1227, 1991.

[2] L. Riha, T. Brzobohaty, A. Markopoulos, "Highly Scalable FETI Methods in ESPRESO", in P. Ivnyi, B.H.V. Topping, (Editors), "Proceedings of the Fourth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering", Civil-Comp Press, Stirlingshire, UK, Paper 17, 2015. doi:10.4203/ccp.107.17

[3] C. Farhat, F-X. Roux: An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems, SIAM J. Sci. Stat. Comput. 13, 379396, 1992.

[4] Farhat, C., Mandel, J., Roux, F-X., *Optimal convergence properties of the FETI domain decomposition method*, Comput. Meth. Appl. Mech. Eng. 115, 365–385, 1994.

[5] Rixen, D. J., Farhat, C., *A simple and effecient extension of a class of substructure based prexonditioners to heterogenous structural mechanics problems*, Int. J. Numer. Meth. Engng. 44, 489–516, 1999.

[6] Z. Dostál, D. Horák, R. Kučera: Total FETI - an easier implementable variant of the FETI method for numerical solution of elliptic PDE, Communications in Numerical Methods in Engineering 22, (12), 1155 1162, 2006.

[7] T. Brzobohatý, Z. Dostál, T. Kozubek, P. Kovář, A. Markopoulos: Cholesky decomposition with fixing nodes to stable computation of a generalized inverse of the stiffness matrix of a floating structure, International Journal for Numerical Methods in Enginering 88, (5), 493 509, 2011. doi:10.1002/nme.3187

[8] Z. Dostál, T. Kozubek, A. Markopoulos, M. Menšík: Cholesky decomposition of a positive semidefinite matrix with known kernel, Applied Mathematics and Computation 217, (13), 60676077, 2011. doi:10.1016/j.amc.2010.12.069

[9] R. Kučera, T. Kozubek, A. Markopoulos: On large-scale generalized inverses in solving two-by-two block linear systems, Linear Algebra and Its Applications 438 (7), 30113029, 2013.

[10] C. Farhat, J. Mandel, F-X. Roux: Optimal convergence properties of the FETI domain decomposition method, Computer Methods in Applied Mechanics and Engineering 115, 365385, 1994.

[11] F-X. Roux, Spectral analysis of interface operator, Proceedings of the 5th Int. Symp. on Domain Decomposition Methods for Partial Differential Equations, ed. D. E. Keyes et al., SIAM, Philadelphia 1992; 73–90.

[12] C. Farhat, J. Mandel, F-X. Roux, Optimal convergence properties of the FETI domain decomposition method, Comput. Methods Appl. Mech. Eng. 115, 1994; 365–385.

[13] F-X. Roux, C. Farhat, Parallel implementation of direct solution strategies for the coarse grid solvers in 2-level FETI method, Contemporary Math. 218, 1998; 158–173.

[14] T. Kozubek, V. Vondrák, M. Menšík, D. Horák, Z. Dostál, V. Hapla, P. Kabelikova, M. Cermak, Total FETI domain decomposition method and its massively parallel implemen tation, Advances of Engeneering Software,

[15] A. Klawonn, O. Rheinbach, Highly scalable parallel domain decomposition methods with an application to biomechanics, ZAMM, 90, No. 1, pp.5-32, 2010

[16] Jungho Lee: A hybrid domain decomposition method and its applications to contact problems in mechanical engineering, PhD thesis, New York University, 2009.

[17] M. Jarošová, T. Kozubek M. Menšík, A. Markopoulos: Hybrid Total FETI method. ECCOMAS 2012, European Congress on Computational Methods in

Applied Sciences and Engineering, e-Book Full Papers, Vienna University of Technology, 2012, p. 6653-6663.