Parallel Computing 57 (2016) 154–166

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Implementation of the efficient communication layer for the highly parallel total FETI and hybrid total FETI solvers

Lubomír Říha*, Tomáš Brzobohatý, Alexandros Markopoulos, Marta Jarošová, Tomáš Kozubek, David Horák, Václav Hapla

IT4Innovations National Supercomputing Center, VŠB-Technical University of Ostrava, Ostrava, Czech Republic

ARTICLE INFO

Article history:
Received 3 November 2014
Revised 15 February 2016
Accepted 12 May 2016
Available online 13 May 2016

Keywords:
FETI
Total FETI
Hybrid total FETI
Domain decomposition
Scalability
HPC

ABSTRACT

This paper describes the implementation, performance, and scalability of our communication layer developed for Total FETI (TFETI) and Hybrid Total FETI (HTFETI) solvers. HTFETI is based on our variant of the Finite Element Tearing and Interconnecting (FETI) type domain decomposition method. In this approach a small number of neighboring subdomains is aggregated into clusters, which results in a smaller coarse problem. To solve the original problem TFETI method is applied twice: to the clusters and then to the subdomains in each cluster.

The current implementation of the solver is focused on the performance optimization of the main CG iteration loop, including: implementation of communication hiding and avoiding techniques for global communications; optimization of the nearest neighbor communication - multiplication with a global gluing matrix; and optimization of the parallel CG algorithm to iterate over local Lagrange multipliers only.

The performance is demonstrated on a linear elasticity 3D cube and real world benchmarks.

1. Introduction

The goal of this paper is to describe parallelization and optimization techniques and algorithms required to implement efficient communication layer for the Finite Element Tearing and Interconnecting (FETI) based parallel solvers. The efficient communication layer is essential for good scalability in the cluster environment. It must be able to run on several thousands of MPI processes and achieve minimal communication overhead. The method that is mainly used for performance evaluation of the communication layer in this paper is the Total FETI with one subdomain per MPI process and one MPI process per CPU core. This configuration uses high number of MPI ranks to solve a problem and therefore relies mainly on the communication layer. This paper also introduces the Hybrid Total FETI (HTFETI) method, which can process several hundreds of small subdomains per MPI process and efficiently run with only one MPI process per node. This means that if TFETI method runs on 400 nodes with 20 MPI processes per node HTFETI will run on 8000 compute nodes.

HTFETI method is based on our variant of the 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 the large linear

* Corresponding author.
E-mail address: lubomir.riha@vsb.cz (L. Říha).

http://dx.doi.org/10.1016/j.parco.2016.05.002
0167-8191/

systems arising in linearized engineering problems by Farhat and Roux [2]. 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 [3] was considered, was efficient only for a small number of subdomains. So it was later extended by introducing a natural coarse problem [4,5], whose solution was implemented by auxiliary projectors so that the resulting algorithm became in a sense optimal [4,5].

In the TFETI method [6], also the Dirichlet boundary conditions are enforced by Lagrange multipliers. Hence all subdomain stiffness matrices are singular with a priori known kernels, which is a great advantage in the numerical solution. With the known kernel basis we can regularize effectively the local stiffness matrix [10] and use any standard Cholesky type decomposition method for nonsingular matrices.

Even if there are several efficient coarse problem parallelization strategies [7], there are still size limitations of the coarse problem. So several hybrid (multilevel) methods were proposed [8,9]. The key idea is to aggregate small number of neighboring subdomains into clusters, which naturally results in smaller coarse problem. In our HTFETI, the aggregation of subdomains into the clusters is enforced again by Lagrange multipliers. Thus the TFETI method is used on both the cluster and subdomain levels. This approach allows parallelization of the original problem up to tens of thousands of cores, which is not reachable with standard FETI methods (difficulties with the large coarse problem). However, convergence of the HTFETI method is slower and therefore the number of iterations is higher when compared to the TFETI method. This means that for smaller problems TFETI remains more efficient and recommended method. But our ultimate goal is to compute extremely large problems decomposed into such a high number of subdomains which are not solvable by the standard FETI methods.

2. Matrix formulation

In this paper, we use the notation introduced in [1]. Let us consider a model problem from linear elasticity. The isotropic elastic body occupies a domain $\Omega \subset \mathbb{R}^d, d = 2, 3,$ with sufficiently smooth boundary Γ. To apply the HTFETI approach to solve such problem, we first of all tear the body from the part of the boundary with the Dirichlet boundary condition as in the TFETI approach. Then we decompose the body into non-overlapping clusters and the clusters into non-overlapping subdomains. Finally, we introduce new gluing conditions on the subdomain interfaces to ensure the continuity of the solution on the boundaries with imposed Dirichlet data.

The resulting quadratic programming (QP) problem after the finite element discretization reads

$$\min_{\mathbf{u}} \frac{1}{2}\mathbf{u}^\top \mathbf{K}\mathbf{u} - \mathbf{f}^\top \mathbf{u} \quad \text{subject to} \quad \mathbf{B}\mathbf{u} = \mathbf{c}, \tag{1}$$

where $\mathbf{K}$ is a block diagonal symmetric positive semidefinite stiffness matrix of size $n \times n$, $\mathbf{f} \in \mathbb{R}^n$ is a load vector, $\mathbf{B} = \left[\mathbf{B}_g^\top,\ \mathbf{B}_D^\top\right]$ denotes an $m \times n$ full-rank equality constraint matrix, and $\mathbf{c} = \left[\mathbf{o}^\top,\ \mathbf{c}_D^\top\right]^\top \in \mathbb{R}^m$ is an equality constraint vector. The matrix $\mathbf{B}_g$ is called a jump operator or a gluing matrix. It is a signed Boolean matrix constructed in such a way that the equality constraint $\mathbf{B}_g\mathbf{u} = \mathbf{o}$ enforces continuity at the subdomain interface degrees of freedom. Satisfaction of this constraint implies that the entries of the solution vector $\mathbf{u}$ are equal if the corresponding interface nodes coincide. The equality constraint $\mathbf{B}_D\mathbf{u} = \mathbf{c}_D$ enforces the Dirichlet boundary conditions. Typically the number of Lagrange multipliers (dual dimension) m is much smaller than n, the number of degrees of freedom (primal dimension). Let us note that $\mathbf{B}$ can be directly assembled to have orthonormal rows only by special treatment of the rows of $\mathbf{B}_g$ corresponding to the nodes shared by more than two subdomains.

The problem (1) has the same structure as in the standard TFETI method and could be solved by this approach. However, to describe the HTFETI method, we will consider the problem (1) in the form

$$\min \frac{1}{2}\mathbf{u}^\top \mathbf{K}\mathbf{u} - \mathbf{u}^\top \mathbf{f} \quad \text{subject to} \quad \begin{cases} \mathbf{B}_0\mathbf{u} = \mathbf{c}_0 \\ \mathbf{B}_1\mathbf{u} = \mathbf{c}_1 \end{cases}, \tag{2}$$

where the equality constraints are split into two parts. The first part $\mathbf{B}_0\mathbf{u} = \mathbf{c}_0 = \mathbf{o}$ consists of m_0 equalities enforcing the continuity in the subdomain corner nodes of each cluster, while $\mathbf{B}_1\mathbf{u} = \mathbf{c}_1$ consists of m_1 equalities enforcing the continuity across the rest of the subdomain interfaces and the Dirichlet conditions, and $m_0 + m_1 = m$.

The optimality conditions for the problem (2) lead to the saddle point problem

$$\left[\begin{array}{ccc} \mathbf{K} & \mathbf{B}_0^\top & \mathbf{B}_1^\top \\ \mathbf{B}_0 & \mathbf{O} & \mathbf{O} \\ \hline \mathbf{B}_1 & \mathbf{O} & \mathbf{O} \end{array}\right] \left[\begin{array}{c} \mathbf{u} \\ \boldsymbol{\lambda}_0 \\ \hline \boldsymbol{\lambda}_1 \end{array}\right] = \left[\begin{array}{c} \mathbf{f} \\ \mathbf{c}_0 \\ \hline \mathbf{c}_1 \end{array}\right] \tag{3}$$

or

$$\left[\begin{array}{c|c} \tilde{\mathbf{K}} & \tilde{\mathbf{B}}^\top \\ \hline [-11pt]\tilde{\mathbf{B}} & \mathbf{O} \end{array}\right]\left[\begin{array}{c} \tilde{\mathbf{u}} \\ \hline [-11pt]\tilde{\boldsymbol{\lambda}} \end{array}\right] = \left[\begin{array}{c} \tilde{\mathbf{f}} \\ \bar{\tilde{\mathbf{c}}} \end{array}\right], \tag{4}$$

where $\tilde{\mathbf{K}}, \tilde{\mathbf{B}}, \tilde{\mathbf{u}}, \tilde{\boldsymbol{\lambda}}, \tilde{\mathbf{f}}, \tilde{\mathbf{c}}$ respect the block structure indicated in (3), $\tilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}_1$, and $\tilde{\mathbf{c}} = \mathbf{c}_1$. The algebraical procedure to solve the system (4) is the same as in the original TFETI scheme (see e.g. [6]), but with different inner structure of the objects, which is emphasized with the tilde. The first equation of the system (4) has a solution if

$$\tilde{\mathbf{f}} - \tilde{\mathbf{B}}^\top\tilde{\boldsymbol{\lambda}} \in \operatorname{Im}\tilde{\mathbf{K}}, \tag{5}$$

which can be expressed more conveniently by means of the kernel $\tilde{\mathbf{R}}$ of the matrix $\tilde{\mathbf{K}}$ as

$$\tilde{\mathbf{R}}^\top(\tilde{\mathbf{f}} - \tilde{\mathbf{B}}^\top\tilde{\boldsymbol{\lambda}}) = \mathbf{o}. \tag{6}$$

Note we equate here and further on a kernel $\mathcal{R}$ of an arbitrary matrix $\mathbf{A}$, and the matrix $\mathbf{R}$ whose columns span the space $\mathcal{R}$, i.e.

$$\mathcal{R} \equiv \mathbf{R} \iff \mathcal{R} = \operatorname{Im}\mathbf{R} = \operatorname{Ker}\mathbf{A}.$$

The matrix $\tilde{\mathbf{R}}$ has the following block structure

$$\tilde{\mathbf{R}} = \begin{bmatrix} \mathbf{R}_c^1 & & \\ & \ddots & \\ & & \mathbf{R}_c^{N_c} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} \end{bmatrix}, \tag{7}$$

where $\mathbf{R}_c^i$ $(i = 1, \ldots, N_c)$ denotes the kernel of the ith cluster, and N_c is the total number of clusters. Furthermore, each cluster kernel consists of subdomain kernels,

$$\mathbf{R}_c^i = \begin{bmatrix} \mathbf{R}_s^{i,1} \\ \vdots \\ \mathbf{R}_s^{i,N_s^i} \end{bmatrix}, \tag{8}$$

with $\mathbf{R}_s^{i,j}$ $(j = 1, \ldots, N_s^i)$ meaning the kernel of the jth subdomain of the ith cluster, and N_s^i standing for the number of subdomains of the ith cluster. Thanks to the TFETI approach, the blocks $\mathbf{R}_s^{i,j}$ may be assembled directly from the rigid body modes of each mesh node. This means each subdomain kernel $\mathbf{R}_s^{i,j}$ consists of segments $\mathbf{R}_p^{i,j,k}$,

$$\mathbf{R}_s^{i,j} = \begin{bmatrix} \mathbf{R}_p^{i,j,1} \\ \vdots \\ \mathbf{R}_p^{i,j,N_p^{i,j}} \end{bmatrix}, \tag{9}$$

where $\mathbf{R}_p^{i,j,k}$ is defined for each mesh node with an index $k = 1, \ldots, N_p^{i,j}$ and spatial coordinates $[x_k, y_k, z_k]$, belonging to the jth subdomain of the ith cluster, as follows

$$\mathbf{R}_p^{i,j,k} = \begin{bmatrix} 1 & 0 & -y_k \\ 0 & 1 & x_k \end{bmatrix} \text{ in 2D},$$

$$\mathbf{R}_p^{i,j,k} = \begin{bmatrix} 1 & 0 & 0 & 0 & -z_k & y_k \\ 0 & 1 & 0 & z_k & 0 & -x_k \\ 0 & 0 & 1 & -y_k & x_k & 0 \end{bmatrix} \text{ in 3D}. \tag{10}$$

In order to eliminate the primal variables $\tilde{\mathbf{u}}$ from the singular system given by the first equation in (4) we use a generalized inverse $\tilde{\mathbf{K}}^+$ of the matrix $\tilde{\mathbf{K}}$, satisfying $\tilde{\mathbf{K}}\tilde{\mathbf{K}}^+\tilde{\mathbf{K}} = \tilde{\mathbf{K}}$. It may be easily verified that if $\tilde{\mathbf{u}}$ is a solution of the first equation of (4), then there exists a vector $\tilde{\boldsymbol{\alpha}}$ such that

$$\tilde{\mathbf{u}} = \tilde{\mathbf{K}}^+(\tilde{\mathbf{f}} - \tilde{\mathbf{B}}^\top\tilde{\boldsymbol{\lambda}}) + \tilde{\mathbf{R}}\tilde{\boldsymbol{\alpha}}. \tag{11}$$

Substituting (11) into the second equation of (4), and using (6), we get the dual formulation

$$\begin{bmatrix} \tilde{\mathbf{F}} & \tilde{\mathbf{G}}^\top \\ \tilde{\mathbf{G}} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\lambda}} \\ \tilde{\boldsymbol{\alpha}} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{d}} \\ \tilde{\mathbf{e}} \end{bmatrix}, \tag{12}$$

where

$$\tilde{\mathbf{F}} = \tilde{\mathbf{B}}\tilde{\mathbf{K}}^{+}\tilde{\mathbf{B}}^\top, \quad \tilde{\mathbf{d}} = \tilde{\mathbf{B}}\tilde{\mathbf{K}}^{+}\tilde{\mathbf{f}} - \tilde{c},$$
$$\tilde{\mathbf{G}} = -\tilde{\mathbf{R}}^\top\tilde{\mathbf{B}}^\top, \quad \tilde{\mathbf{e}} = -\tilde{\mathbf{R}}^\top\tilde{\mathbf{f}}.$$

As in standard TFETI we introduce the orthogonal projector

$$\tilde{\mathbf{P}} = \mathrm{I} - \tilde{\mathbf{G}}^\top(\tilde{\mathbf{G}}\tilde{\mathbf{G}}^\top)^{-1}\tilde{\mathbf{G}} \tag{13}$$

onto the kernel of $\tilde{\mathbf{G}}$. By premultiplying (12) with $\tilde{\mathbf{P}}$ we get

$$\tilde{\mathbf{P}}\tilde{\mathbf{F}}\tilde{\boldsymbol{\lambda}} = \tilde{\mathbf{P}}\tilde{\mathbf{d}} \quad \text{subject to} \quad \tilde{\mathbf{G}}\tilde{\boldsymbol{\lambda}} = \tilde{\mathbf{e}} \tag{14}$$

and after homogenization of the constraints we get

$$\tilde{\mathbf{P}}\tilde{\mathbf{F}}\tilde{\boldsymbol{\lambda}}_{\mathrm{Ker}} = \tilde{\mathbf{P}}(\tilde{\mathbf{d}} - \tilde{\mathbf{F}}\tilde{\boldsymbol{\lambda}}_{\mathrm{Im}}), \tag{15}$$

where $\tilde{\boldsymbol{\lambda}}_{\mathrm{Ker}} \in \mathrm{Ker}\,\tilde{\mathbf{G}}$, $\tilde{\boldsymbol{\lambda}}_{\mathrm{Im}} = \tilde{\mathbf{G}}^\top(\tilde{\mathbf{G}}\tilde{\mathbf{G}}^\top)^{-1}\tilde{\mathbf{e}}$, and $\tilde{\boldsymbol{\lambda}} = \tilde{\boldsymbol{\lambda}}_{\mathrm{Ker}} + \tilde{\boldsymbol{\lambda}}_{\mathrm{Im}}$.

Lemma 1. *The matrix* $\tilde{\mathbf{P}}\tilde{\mathbf{F}}$ *is symmetric positive definite on* $\mathrm{Ker}\,\tilde{\mathbf{G}}$.

Reader can find the proof of Lemma 1 in ([4]) Thanks to the Lemma 1 the problem (15) may be solved efficiently by the PCG (Preconditioned Conjugate Gradients) algorithm. In each PCG iteration we need to compute an action of the operator $\tilde{\mathbf{P}}\tilde{\mathbf{F}}$ which incorporates an action of $\tilde{\mathbf{K}}^{+}$, as follows from the first equation of (12). This means solving a singular system

$$\tilde{\mathbf{K}}\tilde{\mathbf{v}} = \tilde{\mathbf{g}} \iff \begin{bmatrix} \mathbf{K} & \mathbf{B}_0^\top \\ \mathbf{B}_0 & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \boldsymbol{\mu}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{h}_0 \end{bmatrix} \tag{16}$$

which can be carried out using TFETI approach again. For the solution of (16), conditions analogous to (11) and (6), respectively, can be formulated as

$$\exists \boldsymbol{\beta}_0 : \quad \mathbf{v}_0 = \mathbf{K}^{+}(\mathbf{g}_0 - \mathbf{B}_0^\top\boldsymbol{\mu}_0) + \mathbf{R}_0\boldsymbol{\beta}_0, \tag{17}$$

$$\mathbf{R}_0^\top(\mathbf{g}_0 - \mathbf{B}_0^\top\boldsymbol{\mu}_0) = \mathbf{o}, \tag{18}$$

where $\mathbf{R}_0$ is the kernel of $\mathbf{K}$, structured as

$$\mathbf{R}_0 = \begin{bmatrix} \mathbf{R}_{c0}^1 & & \\ & \ddots & \\ & & \mathbf{R}_{c0}^{N_c} \end{bmatrix}, \quad \mathbf{R}_{c0}^i = \begin{bmatrix} \mathbf{R}_s^{i,1} & & \\ & \ddots & \\ & & \mathbf{R}_s^{i,N_s^i} \end{bmatrix}, \quad i = 1, \ldots, N_c. \tag{19}$$

Substituting (17) into the second equation in (16), and using (18) as the new second equation, we get

$$\underbrace{\begin{bmatrix} \mathbf{F}_0 & \mathbf{G}_0^\top \\ \mathbf{G}_0 & \mathbf{O} \end{bmatrix}}_{\mathbf{A}_0} \begin{bmatrix} \boldsymbol{\mu}_0 \\ \boldsymbol{\beta}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{e}_0 \end{bmatrix}, \tag{20}$$

where

$$\mathbf{F}_0 = \mathbf{B}_0\mathbf{K}^{+}\mathbf{B}_0^\top, \quad \mathbf{d}_0 = \mathbf{B}_0\mathbf{K}^{+}\mathbf{g}_0 - \mathbf{h}_0,$$
$$\mathbf{G}_0 = -\mathbf{R}_0^\top\mathbf{B}_0^\top, \quad \mathbf{e}_0 = -\mathbf{R}_0^\top\mathbf{g}_0,$$

Lemma 2. *Let* $\mathbf{K}^{+}$ *be the generalized inverse obtained as the inverse of the regularization* $\mathbf{K}_\rho$ *of the matrix* $\mathbf{K}$, $\mathbf{K}^{+} = \mathbf{K}_\rho^{-1}$ *(introduced in [13]). Then the matrix* $\mathbf{K}^{+}$ *is symmetric positive definite, and so is the matrix* $\mathbf{F}_0$.

The system (20) can be solved in several ways. Because of its small size, it can be explicitly assembled and solved by a direct solver. However, we have to deal with the fact that the matrix $\mathbf{A}_0$ is singular as $\mathbf{G}_0$ is rank-deficient.

One of the possibilities is to introduce a matrix $\bar{\mathbf{R}}_0$, obtained from $\mathbf{R}_0$ by the following procedure: returning to the notation of (19), for each cluster i select its arbitrary subdomain j, and remove from $\mathbf{R}_0$ all columns containing columns of $\mathbf{R}_s^{i,j}$. We introduce objects

$$\bar{\mathbf{G}}_0 = -\bar{\mathbf{R}}_0^\top\mathbf{B}_0^\top, \quad \bar{\mathbf{e}}_0 = -\bar{\mathbf{R}}_0^\top\mathbf{g}_0,$$

where $\bar{\mathbf{G}}_0$ is a full-rank matrix. Using Lemma 2 and replacing $\mathbf{G}_0$ and $\mathbf{e}_0$ by $\bar{\mathbf{G}}_0$ and $\bar{\mathbf{e}}_0$ in (20), we get an SPD system which is then solved instead of (20).

3. Total FETI and hybrid total FETI solver

Our TFETI and HTFETI solver is implemented in pure C++. 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[12] sparse direct solver) of the Intel MKL library [11].

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 that our solver uses. In addition, to be able to port the solver to Intel MIC (in both native or offload mode), the Intel compiler is used for compilation and Intel MPI is used as a message passing library.

The HTFETI method uses a two level decomposition: the problem is decomposed into clusters (the first level), then the clusters are decomposed into subdomains (the second level). In our implementation this decomposition is mapped to a parallel hardware in a following way. The clusters are mapped to compute the nodes of a supercomputer therefore a parallelization model for distributed memory is used - in our case we use the message passing model (MPI). The subdomains inside a cluster are mapped to CPU cores of the particular compute node therefore the shared memory model is used for the second level. Our implementation allows to have multiple clusters associated with a single compute node, but a single cluster cannot be processed on more than one node, as this is a limitation of shared memory parallelization.

There are two major parts of the solver that affect its parallel performance and scalability: (1) a communication layer (described in Section 3.1) and (2) the inter-cluster processing routines (described in Section 3.2). The first part deals with optimization of the TFETI algorithm to minimize its communication overhead caused mainly by gluing matrix ($\mathbf{B}$) multiplication and application of the projector $\mathbf{P}$ (includes multiplication with matrix $\mathbf{G}$ and a coarse problem solution). Having fully optimized communication layer is essential for both the TFETI and HTFETI methods. The second part is a set of specific routines developed for the HTFETI method, which are focused on the efficient parallel processing of multiple subdomains per cluster.

From the implementation point of view the TFETI method can be seen as having a single domain per cluster and using different gluing matrix and different function for application of $\mathbf{K}^+$. The rest of the algorithm remains unchanged.

3.1. Communication layer optimization

The solver uses hybrid parallelization, well suited for multi-socket and multi-core compute nodes, as this is the hardware architecture of most today's supercomputers.

The first level of parallelization is designed for parallel processing of the clusters of subdomains. Each cluster must be assigned to a single node, but if necessary, multiple clusters can be processed per one node. Our implementation does not allow multiple nodes to work on a single cluster. This distributed memory parallelization is done using MPI. In particular, we are using MPI standard 3.0 (implemented in the Intel MPI 5.0 Beta) because the communication hiding techniques implemented in our FETI communication layer require the non-blocking collective operations. In the future we would like to improve the communication layer to use GASPI PGAS communication model.

The essential part of this parallelization is the communication layer. This layer is identical whether the solver runs in the TFETI or HTFETI mode and uses novel communication avoiding and hiding techniques for the main iterative solver. In particular, we have implemented: (1) the Pipelined Conjugate 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 a 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 the global equality constraint matrix action (matrix $\mathbf{B}$ for TFETI and $\mathbf{B}_1$ for HTFETI) – written to employ stencil communication which is fully scalable.

The stencil communication for simple decomposition into four subdomains is shown in Fig. 1 where the Lagrange multipliers (LMs) that connect different neighboring subdomains are depicted in different colors. In each iteration, whenever the LMs are updated, an exchange is performed between the neighboring subdomains to finish the update. This affinity also controls the distribution of the data for the main distributed iterative solver, which iterates over the local LMs only.

In our implementation, each MPI process modifies only elements of the vectors used by the CG solver that match the LMs associated with the particular domain in case of TFETI, or the set of domains in a cluster in case of HTFETI. We call this operation the vector compression. In the preprocessing stage the local part of the gluing matrix is also compressed using the same approach (in this case it means that the empty rows are removed from the matrix) so that we can directly use sparse matrix vector multiplication on the compressed vectors.

ESPRESO library calculates the explicit inverse $(\mathbf{G}\mathbf{G}^T)^{-1}$ in order to reduce the amount of global communication. The inverse matrix is calculated in a distribute fashion to reduce the processing time. The process is as follows. Once the $\mathbf{G}\mathbf{G}^T$ is assembled on the master rank it is broadcasted to all the remaining ranks. Each rank then performs the Cholesky decomposition of the $\mathbf{G}\mathbf{G}^T$ and calculates only one small part of the $(\mathbf{G}\mathbf{G}^T)^{-1}$ that is locally required by the projector $\mathbf{P}$.

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 $\mathbf{I}$ which dimension is equal to dimension of $\mathbf{G}\mathbf{G}^T$. The minor penalty of this method is that all ranks have to perform the Cholesky decomposition. However, this operation is significantly less expensive than several thousand executions of the forward and backward substitution.

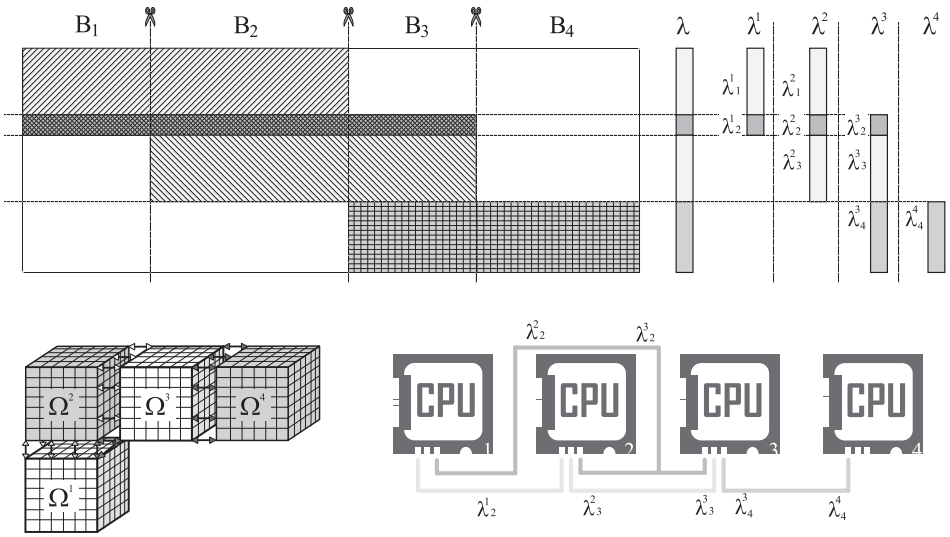

Fig. 1. Stencil communication in TFETI during each action of the gluing matrix $\mathbf{B}_g$.

3.2. Inter-cluster processing

The second level of parallelization is designed for parallel processing of subdomains in a cluster. Our implementation enables over-subscription of CPU cores, therefore each core can process multiple subdomains and the size of the cluster is not limited by the hardware configuration of the compute nodes. If the solver runs in the TFETI mode, this parallelization level is disabled, and to utilize all the CPU cores, multiple MPI processes per node must be executed.

This shared memory parallelization is implemented using Intel Cilk Plus. We have chosen the Cilk Plus 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. We are planning the conversion of the code to OpenMP when standard 4.0 implementation will be released.

The inter-cluster processing can have a large preprocessing time. The most significant part of preprocessing time is the calculation of the matrix $\mathbf{F}_0$ that involves calling the solve routine of the sparse direct solver with many right hand sides for all subdomains in the cluster. Here the right hand side is the matrix $\mathbf{B}_0$ and its size is given by the number of corners. In addition, the performance of the entire routine is affected by the number of domains per cluster and the number of CPU cores per node. For an optimal load-balancing it is necessary that the number of subdomains inside the cluster is in multiples of CPU cores per node.

To explain how necessary the optimal cluster configuration is, we have decomposed two problems of given sizes (in this case 120 000 and 330 000 DOFs) into single cluster and different number of subdomains. The tests were executed on Anselm supercomputer with each node consisting of two 8-core processors. The results are shown in Fig. 2. The highlighted lines (using green color) show the optimal configuration, here the decomposition into 64 subdomains, in terms of shortest iteration time. This decomposition matches perfectly the hardware configuration because each core processes exactly 4 subdomains. In case the preprocessing time needs to be minimized, it is optimal to decompose the problem into 27 subdomains instead of 64. If espreso runs on different cluster with different size of problem, user must searching this optimal decomposition first. We would like to implement routines which do optimal decomposition automatically.

4. Numerical experiments

The described algorithms were implemented in our new library, developed in C++ environment and tested on the solution of 2D and 3D linear elasticity problems. We varied the decomposition and discretization parameters in order to demonstrate the scalability of our method.

The benchmarks were executed on two European supercomputers: (1) Anselm located at IT4Innovations in the Czech Republic and (2) Cartesius located at SurfSara in the Netherlands. The machines have following parameters

- IT4Innovation's Anselm - https://docs.it4i.cz/anselm-cluster-documentation
  - up to 3300 cores
  - non-blocking cluster of 209 nodes each with:
    - 2x 8-core Intel Sandy Bridge E5-2665 (Sandy Bridge), 2.4 GHz and 64 GB of RAM
    - InfiniBand QDR network - 40 Gbit/s inter-node bandwidth

domain size	subdomains	size	avg	sum	prec	iter
4	343	128625	0.123941	5.949182	75.503219	48
5	216	139968	0.072914	3.718598	37.732576	51
6	125	128625	0.040325	2.217873	16.381135	55
8	64	139968	0.030154	1.718762	10.04497	57
11	27	139968	0.034425	1.893399	7.490033	55
17	8	139968	0.064372	3.347334	7.876874	52

Anselm
Problem size:
~120 000

Optimal decomposition:
dom. size: $3*(8+1)^3 = 2187$
cluster size: $4^3 = 64$

domain size	subdomains	size	avg	sum	prec	iter
5	512	331776	0.309891	17.973677	338.035839	58
6	343	352947	0.198848	12.328554	180.041968	62
7	216	331776	0.118965	7.613787	90.402162	64
11	64	331776	0.080177	5.13133	27.456335	64
15	27	331776	0.095755	6.03256	19.761467	63
23	8	331776	0.156146	8.431877	29.455252	54

Anselm
Problem size:
~330 000

Optimal decomposition:
dom. size: $3*(11+1)^3 = 5184$
cluster size: $4^3 = 64$

Fig. 2. Performance of inter-cluster processing for different numbers and sizes of subdomains (Note: domain size N is the number of elements per edge of the cube - real size is equal to $3 \times (N+1)^3$; avg – average iteration time [s]; sum – total time of all iterations = solution time; prec – cluster preprocessing time; iter – number of iterations). (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

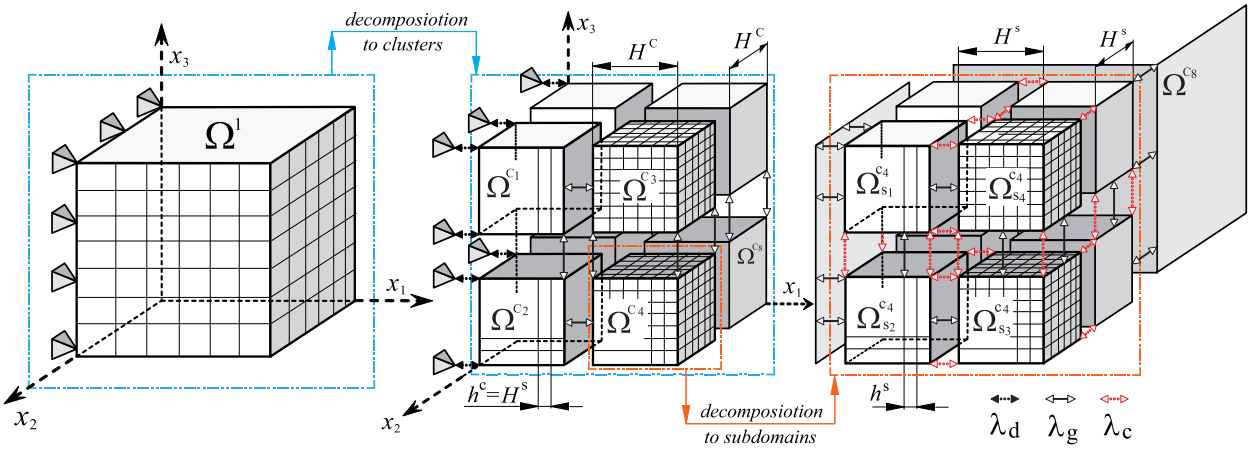

Fig. 3. HTFETI - CUBE benchmark.

- SurfSara's Cartesius - https://surfsara.nl/nl/systems/cartesius
  - up to 8600 cores
  - non-blocking island of 360 nodes each with:
    - 2 12-core Intel Xeon E5-2695 v2 (Ivy Bridge), 2.4 GHz and 64 GB RAM
    - InfiniBand FDR network - 56 Gbit/s inter-node bandwidth

4.1. 3D cube benchmark

Our first benchmark is a linear elasticity problem in a three-dimensional domain. The domain depicted in Fig. 3 has the shape of a cube with the length of the edge 1m. We considered a 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 $E1 = 2.1e5$ [MPa], the Poisson ratio $\nu = 0.3$.

The performance of the solver has been measured in both HTFETI and TFETI to evaluate the scalability of both methods. The focus of this test is to evaluate the time per iteration. Fig. 4 shows the weak scalability of the solver on a 3D cube benchmark where the subdomain size of $3 \cdot (5+1)^3 = 648$ DOFs remains fixed. The small size of the subdomains was intentionally chosen to better observe the behavior of the FETI bottlenecks and the effect of the following optimization methods: (1) Pipelined CG algorithm (in the graph identified as a "CG with 1 reduction"), (2) use of the distributed explicit inverse matrix of the coarse problem $\mathbf{G}\mathbf{G}^\top$ (GGTINV), and (3) use of the HTFETI method. The performance is compared to regular implementation of CG algorithm with two global reductions where the coarse problem is solved on a single master node using parallel sparse MKL PARDISO sparse direct solver (CG with two reductions).

The following observations can be made from the charts: (1) Pipelined CG helps both TFETI and HTFETI method; (2) using GGTINV helps even more for both methods, but we expect that its real potential for the HTFETI method will be seen for

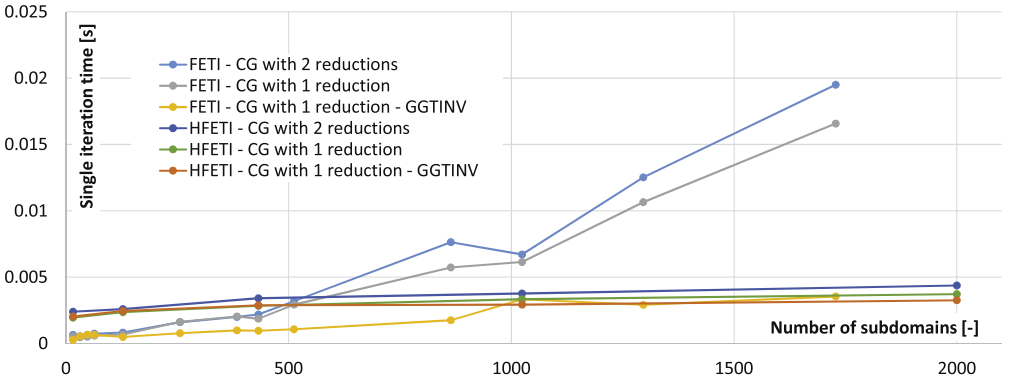

Fig. 4. Comparison of the single iteration time of HTFETI (cluster size 16) and TFETI.

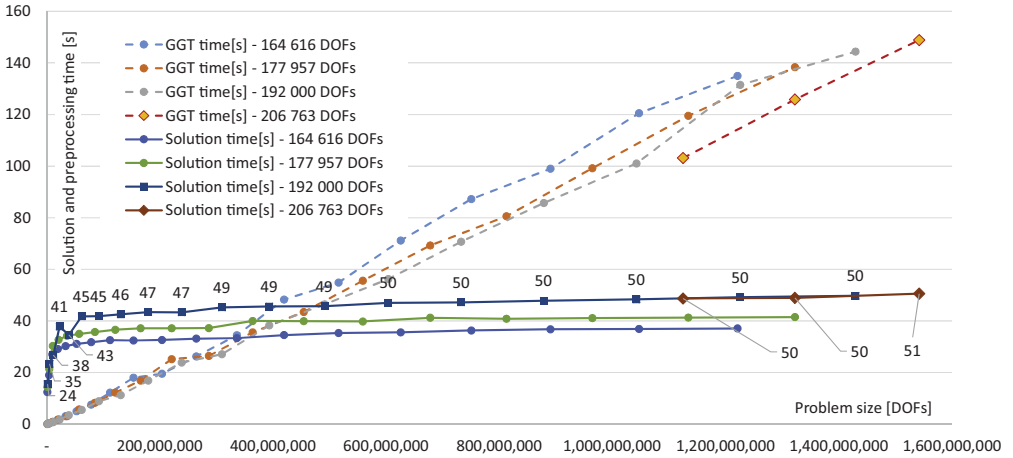

Fig. 5. Weak scalability of large scale benchmarks for up to 8000 subdomains/MPI processes and 4 different subdomain sizes (164,616; 177,957; 192,000; 206,763 DOFs). The figure compares scalability of the coarse problem preprocessing and solution time. The problem size scales up to 1.54 billion DOFs.

large number of clusters; (3) the HTFETI iteration is faster than the TFETI iteration for more than 512 subdomains without GGTINV and for more than 1000 subdomains with GGTINV.

4.2. Large scale benchmark

The main focus of this set of benchmarks is to evaluate the solver behavior for large problems decomposed into a high number of sizeable subdomains, up to 8000, to fully utilize the entire memory resources of a supercomputer. The benchmark is again using the 3D elasticity cube as described in the previous section.

The main focus of the paper is to show the efficient implementation of the communication layer. This layer serves both TFETI and HTFETI method in our solver. However, the TFETI puts significantly higher pressure on this layer, so we used TFETI to show its efficiency. If TFETI scales to 8000 subdomains/MPI processes/343 compute nodes (as presented in this paper), the HTFETI will scale to 8000 clusters, i.e. 8000 compute nodes.

The TFETI solver is used for this testing in order to show the inferior scalability of the coarse problem assembly process which includes these steps: (1) collect the local $\mathbf{G}$ matrix from all MPI processes to the master process, (2) combine these matrices into one global $\mathbf{G}$ matrix, (3) transpose this matrix to create $\mathbf{G}^\top$, (4) calculate the matrix product of $\mathbf{G}$ and $\mathbf{G}^\top$ to create the coarse problem matrix $\mathbf{G}\mathbf{G}^\top$, and (5) broadcast the $\mathbf{G}\mathbf{G}^\top$ matrix to all processes. Each process then calculates its part of the distributed inverse matrix GGTINV, which is a fully parallel task without any communication.

The subdomain sizes are intentionally large to utilize all available memory of every compute node. Four subdomain sizes have been selected for the testing: $3 \cdot (37+1)^3 = 164616$, $3 \cdot (38+1)^3 = 177957$, $3 \cdot (39+1)^3 = 192000$ and $3 \cdot (40+1)^3 = 206763$. Based on the subdomain size, the number of MPI processes per node is 22, 22, 21 and 20, respectively. All measurements shown in Fig. 5 were performed on Cartesius supercomputer.

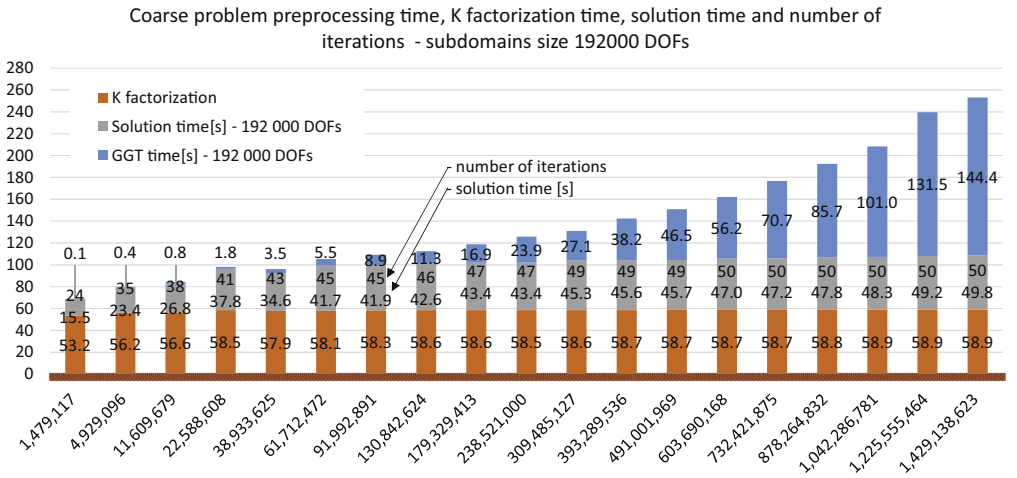

Fig. 6. Total solution time including, (1) the coarse problem $(\mathbf{G}\mathbf{G}^{\top})^{-1}$ preprocessing, (2) factorization of the local stiffness matrix, and (3) solution time for problem decomposed into up 8000 subdomains with size of 192,000 DOFs.

Subdomains of selected sizes produce large stiffness matrices therefore their factorization time represents a significant part of the total processing time. As there is no communication involved in this operation, it does not introduce any parallel overhead. The only parameter that affects the factorization time is the number of MPI processes running per node due to different utilization of each node's memory subsystem.

The next significant part of the total processing time is the solution time. Its scalability is improved using (1) CG algorithm with a single reduction, and (2) distributed inverse matrix of the coarse problem (GGTINV). It can be seen that its scalability is satisfactory and the growth of the solution time is mainly caused by an increasing number of iterations. Please note that for selected subdomain sizes almost 98% of solution time is spent in the solve routine of the sparse direct solver. This means that the application of the coarse problem (GGTINV) and gluing matrix ($\mathbf{B}_1$) in each iteration is almost negligible.

As expected, the coarse problem preprocessing has the worst scalability. It can be seen in Fig. 6 that for a small number of subdomains its execution time is minimal, but for 8000 subdomains the solvers spend more than 57% of the total execution time in this routine.

The largest problems that could be solved with our current version of the TFETI solver using 400 nodes (24 cores and 64 GB RAM per node) of the Cartesius supercomputer were executed in the following configuration: 20 MPI processes per node; 97% of RAM used; subdomain size 206,763 DOFs. In this configuration the solver was able to solve a problem of the size of 1,541,767,203 DOFs (1,654,104,000 DOFs after decomposition) decomposed into 8000 subdomains running on 400 nodes. The processing time of this configuration together with two smaller problem sizes decomposed into 5832 and 6859 subdomains, using 292 and 343 nodes respectively, are shown in Fig. 7. The results show that the solution time is almost identical for two smaller problem sizes (48.7 s and 48.96 s for 50 iterations), and grows for the largest one (50.55s) mainly due to one additional iteration (51). Please note that CG algorithm without preconditioner was used.

4.3. Real world benchmark

The second benchmark is a 2.5 million DOF model of a car engine depicted in Fig. 8. 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, the, 512, and 1024 subdomains in the TFETI mode only on the Anselm supercomputer.

Our first test evaluates various number of MPI processes running per node. Since each Anselm's compute node contains 16 cores, we have run the benchmarks for 8, 15, and 16 MPI processes/subdomains per node. The motivation for this test is as follows. The first case, 8 processes per node, was chosen to observe how doubling the memory bandwidth per core by leaving half of the core unused affects the performance and scalability. The second case, 15 processes per node, leaves one core per node to handle communication routines of the MPI library. In the third case, 16 processes per node, all the CPU cores are used for processing and node is fully utilize by the solver leaving little resources for the system tasks including communication.

The results of this test are shown in Fig. 9. It is important to realize that by running only 8 MPI processes per node we are doubling resources used by the solver when compared to running 16 MPI processes per node. But it can be seen from the results that by using twice as compute nodes the processing time is cut 2.46 times from 0.00476 s to 0.00196 s. We can also see that by leaving one core free for the system tasks the processing time is also reduced, but not as significantly as

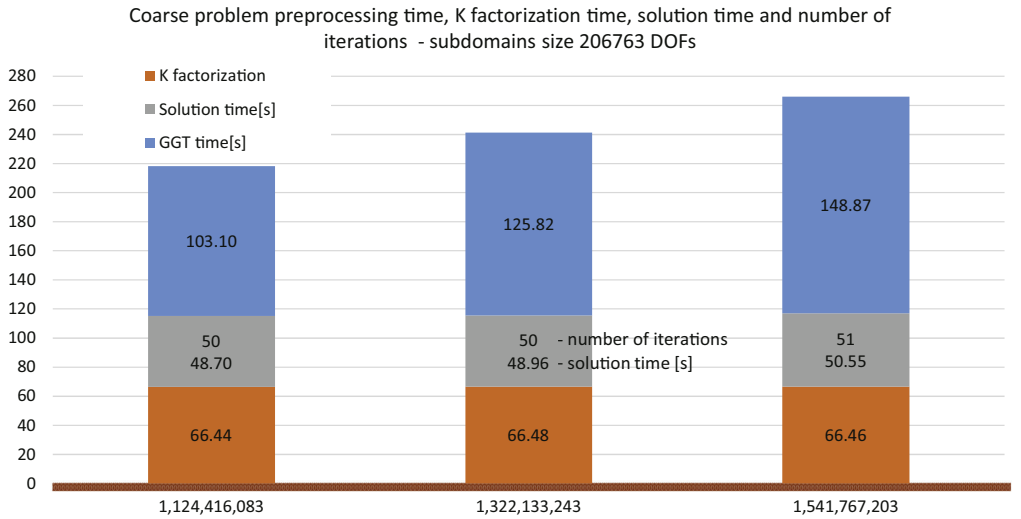

Fig. 7. Total solution time including, (1) coarse problem $(\mathbf{G}\mathbf{G}^{\top})^{-1}$ preprocessing, (2) factorization of the local stiffness matrix, and (3) solution, for problem decomposed into 5832, 6859 and 8000 subdomains with size of 206,763 DOFs.

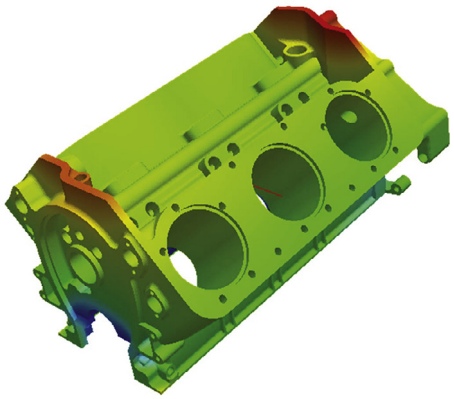

Fig. 8. The car engine benchmark.

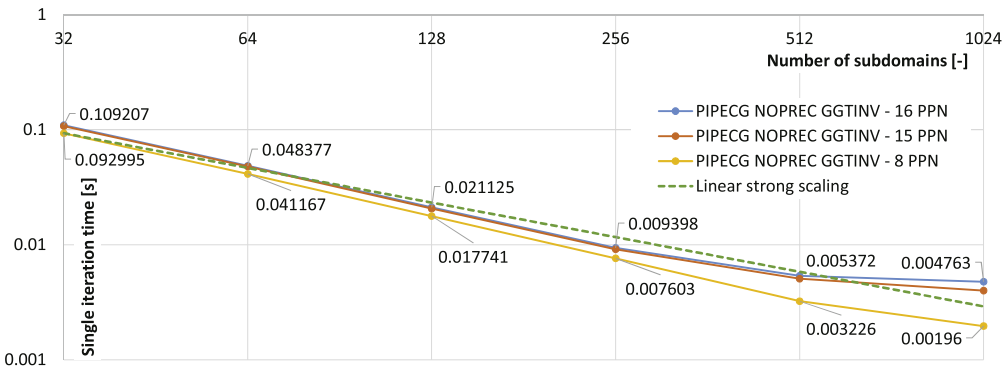

Fig. 9. Numerical scalability evaluation of the real problem "Engine 2.5 Million DOFs" benchmark for the TFETI method. Figure shows the effect of node utilization by running different number of MPI processes per node.

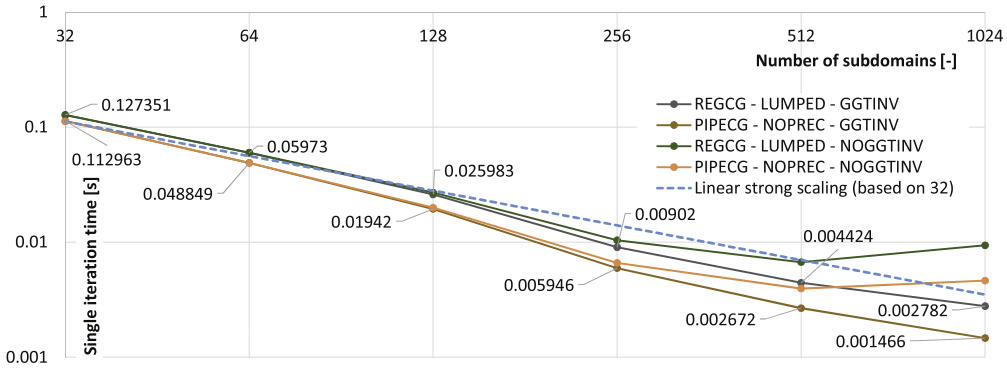

Fig. 10. Strong scaling evaluation of the real problem "Engine 2.5 Million DOFs" benchmark for the TFETI method. Single iteration time for pipelined CG (PIPECG) and regular CG (REGCG) with and without the lumped preconditioner (LUMPED and NOPREC) and distributed inverse matrix of coarse problem (GGTINV and NOGGTINV).

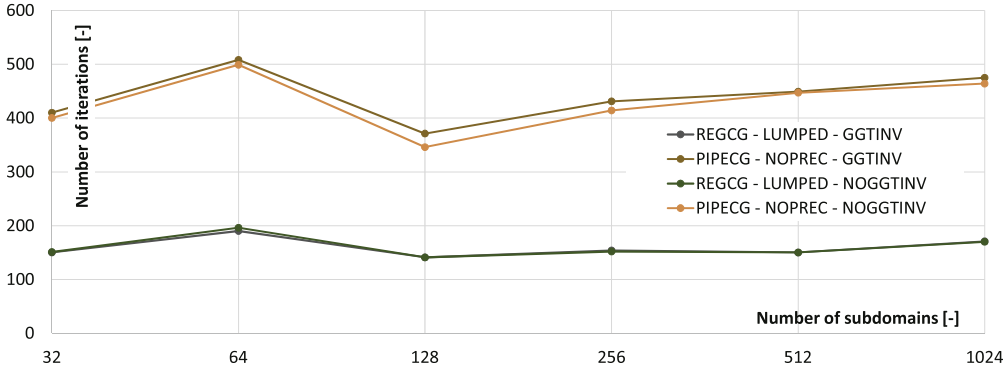

Fig. 11. Strong scaling evaluation of the real problem "Engine 2.5 Million DOFs" benchmark for the TFETI method. Number of iterations for different number of subdomains with and without the lumped preconditioner.

in previous case. Therefore, we are using 8 MPI processes per node configuration for the remaining tests. The TFETI solver works with sparse matrices. As such, it is a memory bounded application. The reason why we used only 8 cores per node out of 16 is to put lower pressure on the memory subsystem. The paper shows that using double the resources (i.e., only half of the resources of a node) the solver runs more than two times faster. The free CPU cores also provide enough resources for the MPI runtime so that the requests by the solver can be server without additional delays. This makes the communication layer more efficient.

Fig. 10 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 the 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 distributed inverse matrix of the coarse problem. As expected, preconditioned regular CG has slower iterations starting from 32 subdomains up to 1024 subdomains. This effect becomes dominant for decomposition into 512 subdomains and essential for the case with 1024 subdomains as the preconditioned regular CG with GGTINV is faster than the pipelined CG without preconditioning. Please note that in both cases using GGTINV keeps the scaling superlinear up to 1024 subdomains. The superlinear effect for strong scaling evaluation is achieved by the reduction of the subdomain sizes and therefore the sizes of their stiffness matrix. This is behavior of the forward and backward substitution operation in the sparse direct solver. Using smaller domains means having higher number of them and therefore it puts higher pressure on the communication layer. If the communication layer is scalable, it can take the superlinear effect of the sparse direct solver and translates it to the superlinear scalability of the entire TFETI solver.

The advantage of using the lumped preconditioner is shown in Fig. 11 where the number of iterations is reduced by 60%–70% on average. When these numbers are combined with the per iteration time we get the entire solution time, shown in Fig. 12. In this figure the significant iteration reduction achieved using the lumped preconditioner is seen from very beginning but gets eliminated by the 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.

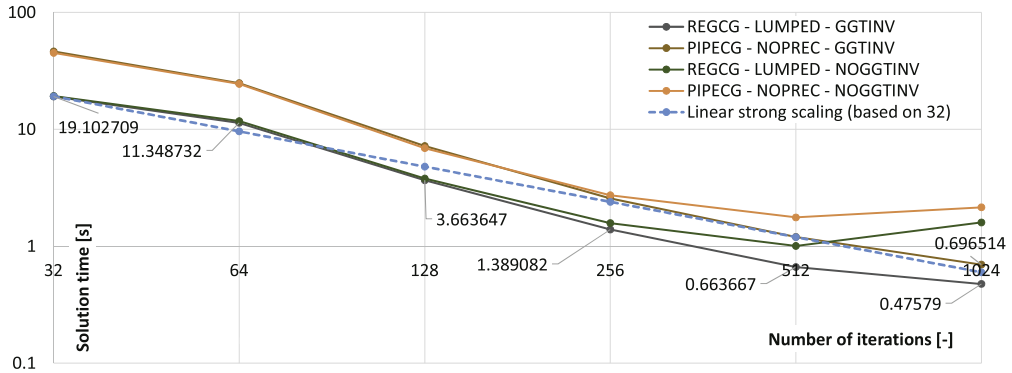

Fig. 12. Strong scaling evaluation of the real problem "Engine 2.5 Million DOF" benchmark for the TFETI method. Entire solution time in seconds for different number of subdomains.

5. Conclusion

The current implementation of the solver is primarily focused on the performance optimization of the main CG solver iteration loop, including: implementation of communication hiding and avoiding techniques for global communications; optimization of the nearest neighbor communication - multiplication with a global gluing matrix; optimization of the parallel CG algorithm to iterate only over local Lagrange multipliers. In other words, we focused on the development of the highly scalable FETI solver. This goal has been successfully achieved as it is shown in Figs. 10 and 12. It can be seen that not only the iteration time but also the solution time scales superlinearly up to 1024 subdomains on a real life benchmark with hundreds of iterations.

We have also proven that our implementation of the TFETI solver can be used to solve large problems of sizes up to 1.6 billion DOF using 400 compute nodes (64 GB of RAM per node) and approximately 25 TB of memory. The memory consumption is mainly caused by large local stiffness matrices and objects generated during their factorizations. These tests show that the main bottleneck for solving large scale problems is the coarse problem preprocessing that cannot be efficiently distributed and it is processed on a single node. Please note that the solver itself scales very well. Its scalability can be shown using an example where problem of various sizes are decomposed into 192,000 DOF subdomains. The iteration time for the problem of the size of 22,588,608 DOF (125 subdomains) is 0.841 s, while for the problem of size 1,429,138,623 (8000 subdomains) it is 0.911 s. This means that for 63 times larger problem the iteration time has been increased only by 8.3%.

We have also implemented the parallel routines for inter cluster processing used by the HTFETI method. The cluster processing is distributed among the CPU cores (shared memory model) only and exploits two levels of parallelism: the data parallelism delivered by the MKL library and the task parallelism achieved using Intel Cilk Plus over multiple subdomains in a cluster. Our evaluation shows the optimal configuration of the clusters in terms of the number of subdomains per cluster and its size to achieve optimal preprocessing and/or iteration time. We have shown that to achieve optimal iteration time the number of subdomains per cluster has to match or be in multiples of the number of CPU cores per node.

Acknowledgment

This paper has been elaborated in the framework of the project called New creative teams in priorities of scientific research, reg. no. CZ.1.07/2.3.00/30.0055, supported by Operational Programme Education for Competitiveness and co-financed by the European Social Fund and the state budget of the Czech Republic and by Grant Agency of the Czech Republic GAČR grant 13-30657P.

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.

The work was supported by the EXA2CT project funded from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 610741.

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

References

[1] 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, pp. 6653–6663.

[2] 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 (1992) 379–396.

[3] F.-X. Roux, et al., Spectral analysis of interface operator, in: D.E. Keyes (Ed.), Proceedings of the 5th International Symposium on Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, 1992, pp. 73–90.

[4] 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.
[5] 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.
[6] Z. Dostál, D. Horák, R. Kučera, Total FETI - an easier implementable variant of the FETI method for numerical solution of elliptic PDE, Commun. Numer. Methods Eng. 22 (12) (2006) 1155–1162.
[7] T. Kozubek, V. Vondrak, M. Mensik, D. Horak, Z. Dostal, V. Hapla, P. Kabelikova, M. Cermak, Total FETI domain decomposition method and its massively parallel implementation, 2013. Advances of Engeneering Software, accepted.
[8] A. Klawonn, O. Rheinbach, Highly scalable parallel domain decomposition methods with an application to biomechanics, ZAMM 90 (No. 1) (2010) 5–32.
[9] J. Lee, A hybrid domain decomposition method and its applications to contact problems in mechanical engineering, New York University, 2009 Phd thesis.
[10] T. Brzobohaty, Z. Dostal, T. Kozubek, P. Kovar, A. Markopoulos, Cholesky decomposition with fixing nodes to stable computation of a generalized inverse of the stiffness matrix of a floating structure, Int. J. Numer. Meth. Engng. 88 (2011) 493–509, doi:10.1002/nme.3187.
[11] Intel Corporation, Intel math kernel library, 2015, https://software.intel.com/en-us/intel-mkl.
[12] O. Schenk, A. Wächter, M. Hagemann, Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization, J. Comput. Optim. Appl. 36 (2-3) (2007) 321–341.