FETI Coarse Problem Parallelization Strategies and Their Comparison

T. Kozubek\textsuperscript{a,∗}, D. Horak\textsuperscript{a}, V. Hapla\textsuperscript{a}

\textsuperscript{a}CE IT4Innovations, VSB-TU of Ostrava, 17. listopadu 15, 70833 Ostrava, Czech Republic

Abstract

Most of computations (subdomain problems) appearing in FETI-type methods are purely local and therefore parallelizable without any data transfers. However, if we want to accelerate also dual actions, some communication is needed due to primal-dual transition. Distribution of primal matrices is quite straightforward. Each of cores works with local part associated with its subdomains. A natural effort using the massively parallel computers is to maximize the number of subdomains so that sizes of subdomain stiffness matrices are reduced which accelerates their factorization and subsequent pseudoinverse application, belonging to the most time consuming actions. On the other hand, a negative effect of that is an increase of the null space dimension and the number of Lagrange multipliers on subdomains interfaces, i.e. the dual dimension, so that the bottleneck of the TFETI method becomes the application of the projector onto the natural coarse space, especially its part called coarse problem solution. In this paper, we suggest and test different parallelization strategies of the coarse problem solution regarding to the improvements of the TFETI massively parallel implementation. Simultaneously we discuss some details of our FLLOP (Feti Light Layer on Petsc) implementation and demonstrate its performance on an engineering elastostatic benchmark of the car engine block up to almost 100 million DOFs. The best parallelization strategy based on the MUMPS was implemented into the multi-physical finite element based opensource code ELMER developed by CSC, Finland.

Keywords Domain decomposition, FETI, TFETI, natural coarse space, coarse problem, parallelization strategies.

1. Introduction

The FETI (Finite Element Tearing and Interconnecting) type methods turned out to be one of the most successful class of methods for parallel solution of elliptic partial differential equations arising from many technical problems. The FETI-1 [3] is based on the decomposition of the spatial domain into non-overlapping subdomains that are ”glued” by Lagrange multipliers. Efficiency of the FETI-1 method was further improved by introducing special orthogonal projectors and preconditioners [4]. The Total-FETI (TFETI) by Dostál et al. [2, 5] simplifies the implementation and the inversion of stiffness matrices of subdomains. Lagrange multipliers are not used only for gluing the subdomains along the auxiliary interfaces, but also to enforce the Dirichlet boundary conditions. This method may be even more efficient than the original FETI-1. The key point is that the null spaces of local stiffness matrices are known a priori, have the same dimension, can be formed directly in parallel, and enable effective regularization of the subdomain stiffness matrices [5, 1].

Naturally one can maximize the number of subdomains so that sizes of subdomain stiffness matrices are reduced. This accelerates their factorization and subsequent pseudoinverse application which belong to the most time consuming actions. A disadvantage of that is an increase of the null space dimension and the dual dimension (the number of Lagrange multipliers), so that for large decompositions the bottleneck of the TFETI method becomes the application of the projector onto the natural coarse space, especially its part called coarse problem solution.

In this paper, we suggest different parallelization strategies of the coarse problem solution and compare them on an engineering benchmark of the car engine block up to 100 millions of DOFs (degrees of freedom). We also discuss some details of our FLLOP (Feti Light Layer on Petsc) implementation concerning computational and programming effectiveness for more complex engineering problems.

∗Corresponding author.

tel. +420 597 323 494 fax. +420 596 919 597 e-mail. tomas.kozubek@vsb.cz
2. FETI/TFETI methods

Let us consider partition of domain $\Omega$ into $N_s$ subdomains $\Omega_s$, $s = 1, \ldots, N_s$ and denote by $K^s$ and $R^s$ the subdomain stiffness matrix and the subdomain null space matrix whose columns span the null space of $K^s$, respectively. Let $B^s$ be a matrix with entries $-1$, $0$, $1$ describing the gluing of subdomains and

$$K = \text{diag}(K^1 \ K^2 \ \ldots \ K^{N_s}), R = \text{diag}(R^1 \ R^2 \ \ldots \ R^{N_s}), B = [B^1 \ B^2 \ \ldots \ B^{N_s}].$$

Let $N_p$ denote the primal dimension, $N_d$ the dual dimension, $N_n$ the null space dimension, $N_s$ the number of subdomains and $N_r$ the number of cores being at disposal for our computation.

The original FETI-1 method assumes that the boundary subdomains inherit the Dirichlet conditions from the original problem, so that the defect of $K^s$ may vary from zero corresponding to the boundary subdomains with sufficient Dirichlet data to fix rigid body motions to the maximum corresponding to the floating subdomains. The basic idea of TFETI [2] is to keep all the subdomains floating and to incorporate the imposed displacements into the matrix of constraints $B$. To implement the boundary conditions like $u_i = 0$, just append the row $b$ to $B$ with all the entries equal to zero except $b_i = 1$. The prescribed displacements will be then enforced by the Lagrange multipliers which may be interpreted as reaction forces. The key point is that the null space matrices $R^s$ are known a priori and they are formed directly in parallel. Moreover, $R^s$ may be exploited for effective regularization of the matrices $K^s$ enabling to use LU and Cholesky type algorithms for nonsingular matrices [5, 1].

Let us apply the duality theory to the primal problem

$$\min \frac{1}{2} u^T Ku - u^T f \quad \text{subject to} \quad Bu = 0$$

and let us consider the following notation

$$F = BK^1 B^T, G = R^T B^T, \bar{G} = TG, \bar{d} = BK^1 f, e = R^T f,$$

where $K^1$ denotes a generalized inverse matrix satisfying $KK^1K = K$, $G$ the natural coarse space matrix, and $G$ its orthonormalization defined by matrix $T (GG^T = I)$. Our minimization problem reads

$$\min \frac{1}{2} \lambda^T F \lambda - \lambda^T \bar{d} \quad \text{s.t.} \quad G\lambda = e.$$  \hspace{1cm} (2)

Further the equality constraints $G\lambda = e$ can be homogenized to $G\lambda = 0$ by choosing arbitrary $\tilde{\lambda}$ which satisfies $G\tilde{\lambda} = e$, e.g. $\tilde{\lambda} = G^T (GG^T)^{-1} e$. Thus substituting $\lambda := \mu + \tilde{\lambda}$ into (2) and denoting $d := \bar{d} - F\tilde{\lambda}$ we get the equivalent problem

$$\min \frac{1}{2} \mu^T PFP \mu - \mu^T Pd \quad \text{s.t.} \quad G\mu = 0,$$

where

$$Q = G^T (GG^T)^{-1} G = \bar{G}^T \bar{G} \quad \text{and} \quad P = I - Q$$

are the orthogonal projectors onto the image space of $G^T$ and onto the null space of $G$, respectively $(\text{Im}Q = \text{Im}G^T$ and $\text{Im}P = \text{Ker}G)$. The problem (3) may be solved effectively by the preconditioned conjugate gradient method with projectors (PCGP) due to classical estimate on the spectral condition number of $PFP$ restricted to the range of $P$ by the ratio of the decomposition parameter $H$ and the discretization parameter $h$ [4]. This remains valid for TFETI.

3. Parallel implementation of FETI/TFETI

Parallelization of FETI/TFETI can be implemented mostly using a data-parallel technique – distributing matrix portions among processing units. This allows algorithms to be almost the same for sequential and parallel case; only the data structure implementation differs. Most of computations (subdomain problems) appearing are purely local and therefore parallelizable without any data transfers. However, if we want to accelerate also dual actions, some communication is needed due to primal-dual transition. Distribution of primal matrices is quite straightforward as every subblock reflects a subdomain - see Figure 1. They can be implemented using general distributed column-block or row-block matrix type with nonzeros only in diagonal blocks. However, $K$ and $R$ possess a nice block-diagonal layout and can be implemented more sophisticatedly using block-diagonal composite type, where subblocks are ordinary sequential matrices and every node holds an array of them. Nevertheless this is not directly implemented in most of parallelization libraries.

Thus each of cores works with local part associated with its subdomains. Current core is specified by rank $(\text{rank} = 0, \ldots, N_s - 1)$. Further some operations require communication through vector transfers. We distinguish three types of these data transfers in the programme:

- elements of the parallel vector are gathered from all cores and summed together into a sequential vector on the master core.
elements of the parallel vector are gathered into identical sequential vectors on all cores,
elements of the vector at certain positions defined by the index set are gathered at positions defined by another index set.

As we mentioned above, a natural approach using the massively parallel computers is to maximize the number of subdomains so that sizes of subdomain stiffness matrices are reduced which accelerates their factorization and subsequent pseudoinverse application, which belong to the most time consuming actions. On the other hand, a negative effect of that is an increase of the null space and dual dimensions, so that the bottleneck of the TFETI method is the application of the projector \( Q \) (see Figure 2).

The natural coarse space matrix \( G \) is computed in a way where each of the cores owns sparse sequential matrices \( R_{\text{rank}} \) and \( B_{\text{rank}} \), so that this core computes the local block \( G_{\text{rank}} = R_{\text{rank}}^T B_{\text{rank}}^T \) of \( G \) matrix without any communication. Redistribution of the horizontal sequential sparse blocks \( G_{\text{rank}} \) into vertical ones is also possible.

Another possibility is not to assemble the matrix \( G \) with advantage concerning the sparsity pattern and fill-in. More precisely, the number of nonzeros in \( G \) matrix is larger than the sum of nonzeros in \( R \) and \( B \) matrices and therefore the matrix-vector and matrix-transpose-vector multiplications are faster using this unassembled form compared to the case with an assembled \( G \) matrix.

Because the actions \( Gv \), \( G^Tv \) take approximately the same time for different \( G \) matrix distributions (assembled \( G \) distributed into horizontal blocks, assembled \( G \) distributed into vertical blocks, unassembled \( G \) kept in the form \( R^T B^T \)), the action time and level of communication depend first of all on implementation of the coarse problem solution

\[
GG^T x = b,
\]

which can be hardly solved sequentially on the master core for large scale problems because of huge memory requirements and loss of scalability.

Within this project, the set of parallelization strategies tested within PRACE-1IP are extended by new strategies based on orthonormalization of \( G \) and exploiting MUMPS [11] library. These strategies are tested and compared on a complex engineering benchmark and their implementation details are discussed in the context of the FLLOP ([13] Feti Light Layer on Petsc) library regarding to computational and programming effectiveness. The machine used for benchmarking is the Hector [12] system at the EPCC site. It consists of the phase 2a (Cray XT5h) machine, the phase 2b (Cray XT6) machine, and an archiving facility. The current HECToR facility is the phase 2b. This was used for our experiments. Phase 2b is a Cray XE6 system offering a total of 1856 XE6 computing nodes. Each compute node contains two AMD 2.1 GHz 12-core processors giving a total of 44,544 cores. Theoretical peak performance is around 373 TFlops. There is presently 32 GB of main memory available per node, which is shared between its twenty-four cores, the total memory is 58 TB. The processors are connected with a high-bandwidth interconnect using Cray Gemini communication chips. The Gemini chips are arranged on a 3 dimensional torus.

4. Parallel implementation of the coarse problem solution

We suggest 4 strategies for the coarse problem solution implying the needed level of preprocessing (see Figure 3(a), gray color is part owned by group of cores in one subcommunicator, dark gray color corresponds to the part owned by each of cores):

1) iteratively using PCG,
2) directly using Cholesky factorization,
3) applying explicit inverse of \( GG^T \),
4) the coarse problem is eliminated, provided that the rows of \( G \) are orthonormalized so that \( (GG^T)^{-1} = I \).
Fig. 3. Different ways of implementation of the coarse problem solution.

The groups of cores - so called subcommunicators - arise from splitting all cores in the global "world" communicator using PETSc built-in pseudopreconditioner PCREDUNDANT specifying by \(N_{\text{red}}\) the number of these subcommunicators (number of cores doing redundant work), i.e. the number of cores in each subcommunicator is equal to \(N_c/N_{\text{red}}\).

In case 1 (see Figures 3(a) and 3(b)) it is necessary to transfer the whole \(G\) matrix to the zeroth core or to all subcommunicators. Master core or subcommunicator cores then compute product \(GG^T\) using matrix-matrix multiplication. The KSP (Krylov subspace methods and preconditioners) iterative solver is then employed for the coarse problem solution sequentially on master core or in parallel in subcommunicators.

In case 2 we have to transfer the whole \(G\) matrix to the zeroth core or to all subcommunicators. Master core or subcommunicator cores then compute product \(GG^T\) using matrix-matrix multiplication. The coarse problem is solved directly given the factorized matrix (by in PETSc built-in Cholesky or LU factorization of \(GG^T\)) sequentially on master core or in parallel using MUMPS [11] in subcommunicators. This second approach has big advantage resulting in the reduction of memory requirements for the coarse problem solution. There are practically no limits because of possible attachments of more and more cores into the subcommunicators.

In case 3 it is necessary to transfer the whole \(G\) matrix to all cores, each of cores then computes sequential product \(GG^T\) using matrix-matrix multiplication. After its factorization we can employ a direct solver for the coarse problem solution to compute the \(i\)th column of \((GG^T)^{-1}\), i.e. inverse of \(GG^T\), on each of cores assigning the \(i\)th column of identity matrix to the RHS vector, so that the inverse can be efficiently computed and applied in parallel if we assign \(N_d/N_c\) corresponding columns of identity matrix to the RHS vector. The coarse problem is then solved by means of ordinary matrix-vector multiplication using the distributed \((GG^T)^{-1}\) matrix.

In case 4 it is necessary to use distribution into vertical blocks for \(G\) matrix. Rows of \(G\) are orthonormalized by the classical Gram-Schmidt algorithm [9] or by forward substitution of factorized \(GG^T\) applied to \(N_d/N_c\) columns of the original matrix \(G\) as RHS \((T = L^{-T}\) with \(L\) being Cholesky factor of coarse problem matrix \(GG^T\)). Another possibility is to apply the external library SLEPC [10] interfacing with PETSc for eigenvalue problems, vector set orthonormalizations, QR factorizations etc. This case has a big advantage - we eliminate the coarse problem \((GG^T = I)\) completely, but we loose the favorable sparsity pattern of the original matrix \(G\).

5. Numerical experiments

Numerical experiments were run on matrices and vectors obtained from decomposition and discretization of complex engineering problem of the car engine block (see Fig. 4). To illustrate both the efficiency of the different strategies of the coarse problem solution and the behavior of the Total FETI we used the decomposition
into 1,014 subdomains and 2.8 million DOFs and then we run the best strategy for the problem with 5,012 subdomains and 98.2 million DOFs as the highlight.

(a) domain decomposition
(b) total displacement

Fig. 4. Car engine benchmark

For cases 1-3 implemented on the master process we get long computational times. Moreover, we are able to solve just small problems, because of master core’s memory limitation while all other cores have to wait until master core finishes its sequential computation.

A big potential of the variant with orthonormalized $G$ matrix and coarse problem elimination was not confirmed. Parallel classical Gram-Schmidt algorithm in SLEPc is too slow and forward solve used for the orthonomalization through Cholesky is again just sequential. Furthermore the orthonormalized matrices are significantly denser and matrix-vector and matrix-transpose-vector multiplications are more time consuming.

In Table 1, we report the computational times for the coarse problem (CP) preprocessing, all coarse problem solutions, all $Q$ actions, and solution time of the PCGP method for our strategies. Obviously the best strategy corresponds to case 2, where the coarse problem is solved using MUMPS in parallel on 14 cores ($N_{red} = 75$) and to case 3, where the coarse problem is solved using explicit inverse in parallel. For those best strategies with respect to the solution time of PCGP the highlight with primal dimension almost 100 million, dual dimension 13.5 million and nullspace dimension 30 thousand is reported in Table 2. We see immediately that the coarse problem preprocessing is much more time consuming in case 3 (explicit inverse) but solution times of PCGP are comparable. Further we see that the number of PCGP iterations increases only moderately from 169 to 176 while the problem size increases from 2.8 to 98.2 million DOFs which illustrates numerical scalability.

<table>
<thead>
<tr>
<th>times [sec]</th>
<th>CP preproc.</th>
<th>CP solutions</th>
<th>$Q$ actions</th>
<th>solution time</th>
<th>per 1 iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (KSP PCG, $N_{red}=16, 1e-08$)</td>
<td>0.12</td>
<td>493</td>
<td>498</td>
<td>493</td>
<td>2.54</td>
</tr>
<tr>
<td>1 (KSP PCG, $N_{red}=1,014,1e-08$)</td>
<td>0.09</td>
<td>39.3</td>
<td>42.3</td>
<td>47.6</td>
<td>0.255</td>
</tr>
<tr>
<td>2 (PETSc LU, master)</td>
<td>2.8</td>
<td>2.29</td>
<td>3.81</td>
<td>11.1</td>
<td>0.06</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=5$)</td>
<td>1.8</td>
<td>3.95</td>
<td>6.72</td>
<td>11.6</td>
<td>0.07</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=10$)</td>
<td>0.71</td>
<td>2.77</td>
<td>5.70</td>
<td>10.7</td>
<td>0.06</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=15$)</td>
<td>0.53</td>
<td>2.42</td>
<td>5.32</td>
<td>10.2</td>
<td>0.06</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=30$)</td>
<td>0.36</td>
<td>2.23</td>
<td>5.21</td>
<td>10</td>
<td>0.059</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=50$)</td>
<td>0.43</td>
<td>2.82</td>
<td>5.50</td>
<td>12</td>
<td>0.06</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=65$)</td>
<td>0.62</td>
<td>2.48</td>
<td>5.30</td>
<td>10</td>
<td>0.06</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=75$)</td>
<td>0.88</td>
<td>2.18</td>
<td>5.39</td>
<td>9.96</td>
<td>0.059</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=100$)</td>
<td>1.3</td>
<td>2.49</td>
<td>5.30</td>
<td>9.99</td>
<td>0.059</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=250$)</td>
<td>2.5</td>
<td>3.49</td>
<td>6.60</td>
<td>11.1</td>
<td>0.066</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=500$)</td>
<td>4.9</td>
<td>9.15</td>
<td>12</td>
<td>16</td>
<td>0.094</td>
</tr>
<tr>
<td>2 (MUMPS, $N_{red}=1,014$)</td>
<td>7.1</td>
<td>26.3</td>
<td>29.1</td>
<td>34.1</td>
<td>0.201</td>
</tr>
<tr>
<td>3 (Explicit)</td>
<td>3.3</td>
<td>2.24</td>
<td>4.9</td>
<td>9.26</td>
<td>0.055</td>
</tr>
<tr>
<td>4 (Chol.)</td>
<td>23</td>
<td>0.00</td>
<td>31.1</td>
<td>34.6</td>
<td>0.201</td>
</tr>
<tr>
<td>4 (SLEPC-CGS)</td>
<td>140</td>
<td>0.00</td>
<td>9.72</td>
<td>12.2</td>
<td>0.074</td>
</tr>
</tbody>
</table>

6. Conclusions

For the used architecture (HECToR) and our PETSc-based implementation (FLLOP) we recommend to use parallel direct solver (MUMPS) for the efficient coarse problem solution of large scale engineering problems.
Table 2. Performance of the coarse problem solution for two best strategies, $N_p=98,214,558$; $N_d=13,395,882$; $N_n=30,072$; $N_c=N_s=5,012$; CG iterations: 176

<table>
<thead>
<tr>
<th>times [sec]</th>
<th>CP preproc.</th>
<th>CP solutions</th>
<th>Q actions</th>
<th>solution time per 1 iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (MUMPS, $N_{red}=75$)</td>
<td>3.2</td>
<td>77.9</td>
<td>110</td>
<td>217</td>
</tr>
<tr>
<td>3 (Explicit)</td>
<td>80</td>
<td>68.9</td>
<td>100</td>
<td>219</td>
</tr>
</tbody>
</table>

There is also a chance for the case with explicit inverse to reduce both memory and computational costs by means of MUMPS used for factorizations in subcommunicators. Another possibility how to reduce the coarse problem size is to use a hybrid (multilevel) FETI method. The best parallelization strategy based on the MUMPS was implemented into multi-physical finite element based opensource code ELMER developed by CSC, Finland [14].

Acknowledgements

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and by the project SPOMECH - Creating a multidisciplinary R&D team for reliable solution of mechanical problems, reg. no. CZ.1.07/2.3.00/20.0070 within Operational Programme ‘Education for competitiveness’ funded by Structural Funds of the European Union and state budget of the Czech Republic. The research has also been supported by the grants: The Czech Science Foundation GACR 101/09/P601, Ministry of Education, Youth and Sports of the CR No. MSM 6198910027. HPC-Europa2 project funded by the European Commission - DG Research in the Seventh Framework Programme under grant agreement No. 228398 and PRACE-2IP project receiving funding from the EU’s Seventh Framework Programme (FP7/2007-2013) under grant agreement No. RI-283493.

References