Contemporary Mathematics
Volume **218**, 1998
B 0-8218-0988-1-03024-7

An Efficient FETI Implementation on Distributed Shared Memory Machines with Independent Numbers of Subdomains and Processors

Michel Lesoinne and Kendall Pierson

1. Introduction

Until now, many implementations of the FETI method have been designed either as sequential codes on a single CPU, or as parallel implementations with a *One Subdomain per Processor* approach. This approach has been particularly typical of implementations on distributed memory architectures such as the IBM SP2. In the last couple of years, several computer manufacturers have introduced new machines with a Distributed Shared Memory (DSM) programming model –e.g. SGI Origin 2000, or HP Exemplar. In such architectures, the physical memory is distributed among the processors or CPU boards but any memory location can be accessed logically by any CPU independently of where the particular memory page being accessed has physically been allocated. As more and more machines of this type are available with a relatively small number of processors, the interest in implementing FETI with an independent number of subdomains and processor has increased. We report on such an implementation of FETI and highlight the benefits of this feature. We have found that medium size to large problems can be solved even on a sequential machine with time and memory requirements that are one to two order of magnitude better than a direct solver.

2. Objectives

When writing our new FETI code, the main objectives were:

- Efficient data structures for distributed shared memory
- Number of subdomains independent of the number of processors

The second requirement was the most important requirement and, when taken to the extreme of a single processor, naturally leads to being able to run the same code sequentially

---

1991 *Mathematics Subject Classification.* Primary 65Y05; Secondary 65N55, 65Y10.
The first author acknowledges partial support by ANSYS, Inc.

3. Overview of FETI

In order to keep this paper self-contained as much as possible, we begin with an overview of the original FETI method [**6, 1, 2, 4, 5**].

The problem to be solved is

$$Ku = F \tag{1}$$

where K is an $n \times n$ symmetric positive semi-definite sparse matrix arising from the finite element discretization of a second- or fourth-order elastostatic (or elastodynamic) problem defined over a region Ω, and F is a right hand side n-long vector representing some generalized forces. If Ω is partitioned into a set of N_s *disconnected* substructures $\Omega^{(s)}$, the FETI method consists in replacing Eq (1) with the equivalent system of substructure equations

$$\begin{aligned} K^{(s)}u^{(s)} &= F^{(s)} - {B^{(s)}}^T\lambda \qquad s = 1, \ ..., \ N_s \\ \Delta &= \sum_{s=1}^{N_s} B^{(s)}u^{(s)} = 0 \end{aligned} \tag{2}$$

where $K^{(s)}$ and $F^{(s)}$ are the unassembled restrictions of K and F to substructure $\Omega^{(s)}$, λ is a vector of Lagrange multipliers introduced for enforcing the constraint $\Delta = 0$ on the substructure interface boundary $\Gamma_I^{(s)}$, and $B^{(s)}$ is a signed Boolean matrix that describes the interconnectivity of the substructures. A more elaborate derivation of (2) can be found in [**6, 3**] In general, a mesh partition may contain $N_f \leq N_s$ floating substructures — that is, substructures without enough essential boundary conditions to prevent the substructure matrices $K^{(s)}$ from being singular — in which case N_f of the local Neumann problems

$$K^{(s)}u^{(s)} = F^{(s)} - {B^{(s)}}^T\lambda \quad s = 1, \ ..., \ N_f \tag{3}$$

are ill-posed. To guarantee the solvability of these problems, we require that

$$(F^{(s)} - {B^{(s)}}^T\lambda) \perp \text{Ker } (K^{(s)}) \quad s = 1, \ ..., \ N_f \tag{4}$$

and compute the solution of Eq. (3) as

$$u^{(s)} = {K^{(s)}}^+(F^{(s)} - {B^{(s)}}^T\lambda) + R^{(s)}\alpha^{(s)} \tag{5}$$

where ${K^{(s)}}^+$ is a generalized inverse of $K^{(s)}$ that needs not be explicitly computed [**4**], $R^{(s)} = \text{Ker } (K^{(s)})$ is the null space of $K^{(s)}$, and $\alpha^{(s)}$ is a vector of six or fewer constants. The introduction of the additional unknowns $\alpha^{(s)}$ is compensated by the additional equations resulting from (4)

$${R^{(s)}}^T(F^{(s)} - {B^{(s)}}^T\lambda) = 0 \quad s = 1, \ ..., \ N_f \tag{6}$$

Substituting Eq. (5) into the second of Eqs. (2) and using Eq. (6) leads after some algebraic manipulations to the following FETI interface problem

$$\begin{bmatrix} F_I & -G_I \\ -{G_I}^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \alpha \end{bmatrix} = \begin{bmatrix} d \\ -e \end{bmatrix} \tag{7}$$

where

$$
\begin{aligned}
F_I &= \sum_{s=1}^{N_s} B^{(s)} K^{(s)^+} B^{(s)^T}; \\
G_I &= \left[B^{(1)} R^{(1)} \quad ... \quad B^{(N_f)} B^{(N_f)} \right]; \\
\alpha^T &= \left[\alpha^{(1)^T} \quad ... \quad \alpha^{(N_f)^T} \right]; \\
d &= \sum_{s=1}^{N_s} B^{(s)} K^{(s)^+} F^{(s)}; \\
e^T &= \left[F^{(1)^T} B^{(1)} \quad ... \quad F^{(N_s)^T} B^{(N_f)} \right]
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
K^{(s)^+} &= K^{(s)^{-1}} \quad \text{if } \Omega^{(s)} \text{ is not a floating substructure} \\
K^{(s)^+} &= \text{a generalized inverse of } K^{(s)} \text{ if } \Omega^{(s)} \text{ is a floating substructure}
\end{aligned}
\tag{9}
$$

For structural mechanics and structural dynamics problems, F_I is symmetric because the substructure matrices $K^{(s)}$ are symmetric. The objective is to solve by a PCG algorithm the interface problem (7) instead of the original problem (1). The PCG algorithm is modified with a projection enforcing that the iterates λ_k satisfy (6). Defining the projector P using

$$
P = I - G_I (G_I^T G_I)^{-1} G_I^T \tag{10}
$$

the algorithm can be written as:

$$
\begin{aligned}
&\text{1. Initialize} \\
&\lambda^0 = G_I \ ({G_I}^T G_I)^{-1} e \\
&r^0 = d - F_I \lambda^0 \\
&\text{2. Iterate } k = 1,\ 2,\ \text{... until convergence} \\
&w^{k-1} = P^T r^{k-1} \\
&z^{k-1} = \overline{F_I^{-1}} \bar{w}^{k-1} \\
&y^{k-1} = P\ \bar{z}^{k-1} \\
&\zeta^k = {y^{k-1}}^T w^{k-1} / {y^{k-2}}^T w^{k-2} \quad (\zeta^1 = 0) \\
&p^k = y^{k-1} + \zeta^k p^{k-1} \quad (p^1 = y^0) \\
&\nu^k = {y^{k-1}}^T w^{k-1} / {p^k}^T F_I p^k \\
&\lambda^k = \lambda^{k-1} + \nu^k p^k \\
&r^k = r^{k-1} - \nu^k F_I p^k
\end{aligned}
\tag{11}
$$

4. Data organization on DSM computer architecture

To be able to efficiently organize data for the FETI solver, we need to examine how the operating system will distribute memory inside the physical memory units and how this distribution affects the cost of accessing that memory. Simple observations of the impact of the computer architecture will give us guidelines to organize the elements involved in the FETI solver. The single distributed shared memory model of DSM machines simplifies the writing of parallel codes. However programmers must be conscious that the cost of accessing memory pages is not uniform and depends on the actual location in hardware of a page being accessed. On the SGI Origin 2000 machine, the operating systems distributes pages of memory onto physical pages by a *first touch* rule. This means that if possible, a memory page is allocated in the local memory of the first CPU that touches the page.

Fortunately, the FETI method, because it is decomposition based, lends itself in a natural way to a distribution of data across CPUs that guarantees that most of the memory is always accessed by the same CPU. To achieve this, one simply applies a distributed memory programming style on a shared memory architecture. This means that all operations relative to a subdomain s are always executed by the same CPU. This way, such objects as the local stiffness matrix $K^{(s)}$ will be created, factored and used for resolution of linear systems always on the same CPU.

5. Parallel programming paradigm

One can easily see that the operations involved in the FETI method are mostly matrix and vector operations that can be performed subdomain-per-subdomain. Such quantities as the residual vector or search direction vectors can be thought of as global quantities made of the assembly of subvectors coming from each subdomain. Operations on such vectors such as sum or linear combinations can be performed on a subdomain per subdomain basis. On the other hand, coefficients such as ν_k and ζ_k are scalar values which are global to the problem. These coefficients ensue mostly from dot products. Dot products can be performed by having the dot product of the subparts of the vectors performed by each subdomain in parallel, and then all the contributions summed up globally.

Such remarks have led us to write the program with a single thread executing the main PCG loop. In that way, only one CPU allocates and computes the global variables such as ν_k and ζ_k. To perform parallel operations, this single thread creates logical *tasks* to be performed by each CPU, and these tasks are distributed to as many parallel threads as CPUs being used. Examples of tasks are the assembly or factorization of $K^{(s)}$, or the update of the subpart of a vector for subdomain s.

Because the number of threads is arbitrary and independent of the number of tasks to be performed at a given step — i.e. several tasks can be assigned to the same thread —, independence between the number of subdomains and the number of CPUs is trivially achieved. In the extreme case, all tasks are executed by a single thread — the main thread — and the program can run on a sequential machine.

6. Implementation of the projector P

The application of the projector P is often referred to as the *coarse problem* because it couples all subdomains together. The application of the projector to a vector z can be seen as a three step process:

- Compute $\gamma = G^t z$
- Solve $(G^t G)\alpha = \gamma$
- Compute $y = z - G\alpha$

The first and last operation can be obtained in parallel with a subdomain per subdomain operation, since the columns of G (the trace of the rigid body modes) are non zero only for one subdomain interface. The second operation however is a global operation that couples all subdomains together.

Past implementations of the projector at the University of Colorado have relied on an iterative solution of the second equation of Eqs (6). Though such an approach was justified by the fact that these implementations were targeted at distributed memory machines, an experimental study of the problem has revealed that on DSM machine, it is more economical to solve this system with a direct solver. This direct solver can only be parallelized for a low number of CPU before losing performance.

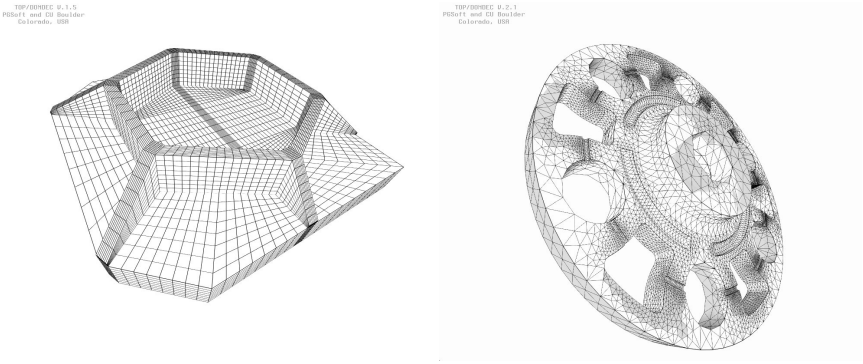

FIGURE 1. Finite element models of a lens (left) and a wheel carrier (right)

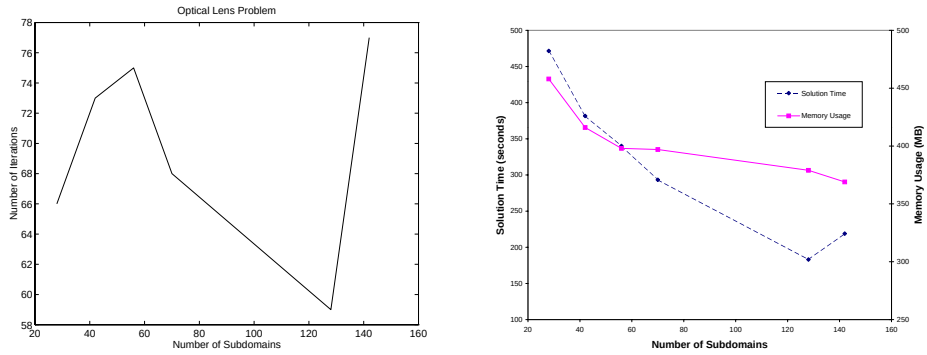

FIGURE 2. Number of iterations, solution time and memory requirements vs number of subdomains (Lens problem, 1 CPU)

Consequently, it is the least parallelizable part of the FETI solver and sets an upper limit to the performance that can be attained by adding CPUs.

7. Numerical experimentations

We have run our code on two significant example problems. For each problem we made runs with a varying number of subdomains and have recorded both timing of the solution and the memory required by the solver.

7.1. Lens problem. The first problem, shown in Fig. 1 on the left has 40329 nodes, 35328 brick elements and 120987 degrees of freedom. Solving this problem sequentially using a skyline solver and renumbered using RCM uses 2.2GB of memory and 10,000 seconds of CPU time. By contrast, on the same machine, running FETI sequentially with 128 subdomains requires 379MB of memory and 183.1 seconds of CPU. This results dramatically highlights that FETI is an excellent solution method, even on sequential machines. As can be seen on the left of Fig. 2, the number of iterations remains stable as the number of subdomains increases. The right part of the same figure shows the variation of timings and memory usage

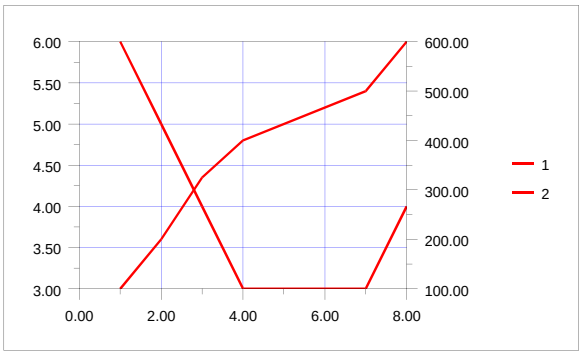

FIGURE 3. Solution time vs number of processors (lens problem, 128 subdomains)

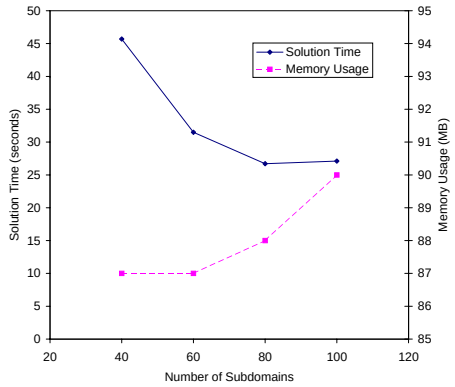

FIGURE 4. Solution time and memory requirements vs number of subdomains (Wheel carrier problem, 1CPU)

with the number of subdomains. It can be noticed that as the number of subdomains increases, the memory required to store all the local stiffness matrices $K^{(s)}$ decreases. This is mainly due to the reduction of the bandwidth of the local problems. Our experimentations show that the optimum decomposition for CPU time has 128 subdomains. Such a number would make it impractical for most users to use FETI with an implementation that requires one CPU per subdomain. After determining the optimum number of subdomains, we ran the same test case with an increasing number of processors. The resulting timings show good scalability (Fig. 3)

7.2. Wheel carrier problem. The second problem is a wheel carrier problem (see Fig. 1 on the right) with 67768 elements, 17541 nodes and 52623 degrees of freedom. The skyline solver requires 576 MB of memory and 800 seconds of CPU time. Fig. 4 shows a single CPU performance of FETI with 80 subdomains of 26.7s. On this problem, the memory requirement beyond 40 subdomains is stable around 88MB but tends to slightly increase as the number of subdomains increases. This is explained by the fact that as the number of subdomains increases, the size of the

interface increases and the memory required to store larger interface vectors offsets the reduction of memory required by the local stiffness matrices $K^{(s)}$.

8. Conclusions

We have implemented the FETI method on Distributed Shared Memory machines. We have achieved independence of the number of subdomains with respect to the number of CPUs. This independence has allowed us to explore the use of a large number of CPUs for various problems. We have seen from this experimentation that using a relatively large number of subdomains (around 100) can be very beneficial both in solution time and in memory usage. With such a high number of subdomains, the FETI method was shown to require CPU times and memory usage that are almost two orders of magnitude lower than those of a direct skyline solver. This strongly suggests that the FETI method is a viable alternative to direct solvers on medium size to very large scale problems.

References

1. C. Farhat, *A Lagrange multiplier based divide and conquer finite element algorithm*, J. Comput. Sys. Engrg. **2** (1991), 149–156.
2. C. Farhat, *A saddle-point principle domain decomposition method for the solution of solid mechanics problems*, Proc. Fifth SIAM Conference on Domain Decomposition Methods for Partial Differential Equations (D.E. Keyes, T.F. Chan, G.A. Meurant, J.S. Scroggs, and R.G. Voigt, eds.), SIAM, 1991, pp. 271–292.
3. C. Farhat, J. Mandel, and F.X. Roux, *Optimal convergence properties of the FETI domain decomposition method*, Comput. Meths. Appl. Mech. Engrg. **115** (1994), 367–388.
4. C. Farhat and F.X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, Internat. J. Numer. Meths. Engrg. **32** (1991), 1205–1227.
5. ———, *An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems*, SIAM J. Sc. Stat. Comput. **13** (1992), 379–396.
6. ———, *Implicit parallel processing in structural mechanics*, Computational Mechanics Advances **2** (1994), 1–124.

Department of Aerospace Engineering and Sciences and Center for Aerospace Structures University of Colorado at Boulder Boulder, CO 80309-0429, U.S.A.