

Some Ideas to Parallelization of Hermes

Pavel Solin

hp-FEM group, University of Nevada, Reno
Academy of Sciences, Prague, Czech Republic

<http://hpfem.org>

July 2009

Abstract

This is a simple manual describing how to parallelize a FEM code using a direct solver. The ideas are widely known in the domain decomposition community, here I just present them in a user-friendly fashion. Parallelization is especially appealing in 3D, since it is the only way to handle very large problems with a direct solver.

1 Weak formulation

Let us consider a setting displayed in Fig. 1, where a 2D domain is decomposed into four non-overlapping subdomains.

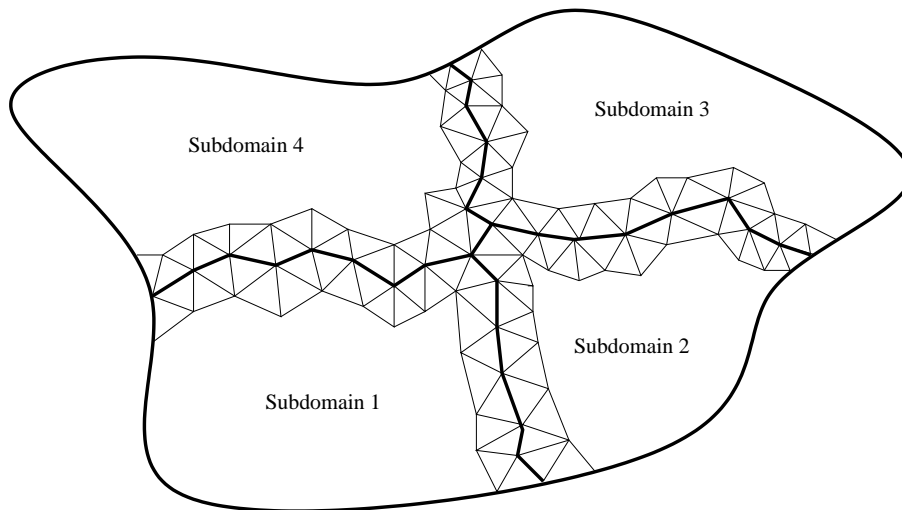


Figure 1: Decomposition of a sample domain.

We assume a finite element mesh over the entire domain but show only one element layer along the interfaces, since this layer has a special importance. More precisely, this set contains all elements that have at least one vertex on the interface.

By S_1, S_2, S_3, S_4 we denote the sets of basis functions whose supports are internal to the first, second, third and fourth subdomain, respectively. By S_I we will mean the set of all basis functions whose supports lie in two or more subdomains. In 2D, these are all vertex and edge functions corresponding to any vertex or mesh edge lying on the interface. In 3D,

there will be also face functions corresponding to element faces lying on the interface. The sum of elements of the sets S_1, S_2, S_3, S_4 and S_5 is equal to the dimension of the FE space, i.e., to the size of the stiffness matrix.

By M_1, M_2, M_3, M_4 we denote the stiffness matrices corresponding to the first, second, third, and fourth subdomain, respectively. The boundary conditions are *homogeneous Dirichlet* on the internal interface, and they remain unchanged on the original domain boundary. The global stiffness matrix consists of four blocks as shown in Fig. 2.

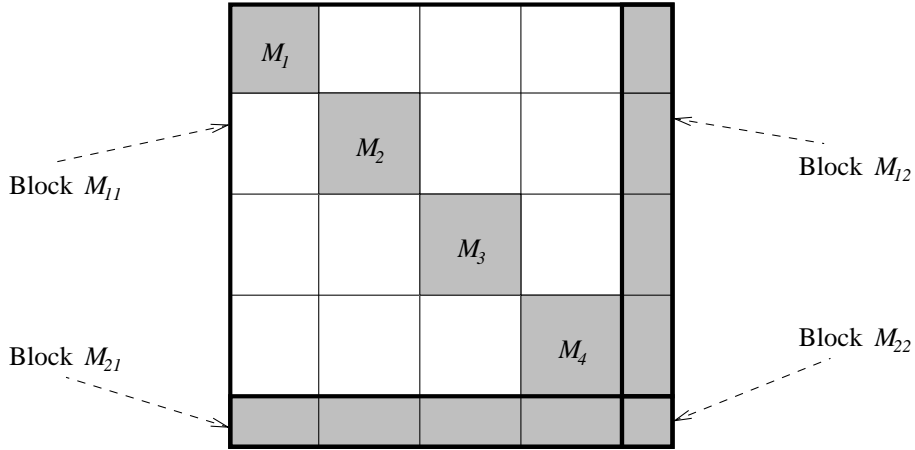


Figure 2: Stiffness matrix structure.

The grey color denotes blocks where nonzeros occur. The off-diagonal blocks in the block M_{11} only contain zeros. Note that the blocks M_1, M_2, M_3, M_4 are disjoint. We will assume that all of them are invertible, and that the block M_{22} (containing products of interfacial degrees of freedom with themselves) is invertible as well.

Since the block M_{11} is easily invertible, we will employ the Schur complement method. The global matrix problem can be written as

$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}. \quad (1)$$

Here, Y_1 is the vector of unknowns corresponding to internal degrees of freedom in the four subdomains and Y_2 comprises coefficients corresponding to the interfacial degrees of freedom. Analogously, the right-hand side vectors F_1, F_2 correspond to internal and interfacial degrees of freedom, respectively. Matrix equation (1) can be written as

$$\begin{aligned} M_{11}Y_1 + M_{12}Y_2 &= F_1. \\ M_{21}Y_1 + M_{22}Y_2 &= F_2. \end{aligned}$$

From here we obtain a system for Y_2 ,

$$(M_{22} - M_{21}M_{11}^{-1}M_{12})Y_2 = F_2 - M_{21}M_{11}^{-1}F_1. \quad (2)$$

Note that the size of this system is equal to the number of interfacial degrees of freedom (small system). After solving it, we obtain Y_2 and use it to calculate Y_1 via

$$Y_1 = M_{11}^{-1}(F_1 - M_{12}Y_2). \quad (3)$$

The solution algorithm itself depends on the parallel architecture we use. In any case, however, we need to construct the LU decomposition of the blocks M_1 , M_2 , M_3 and M_4 – this is done in parallel and in the case of distributed memory, the LU decompositions are stored locally at the processors. Note that the Schur complement method requires repeated solution of a matrix system with the matrix M_{11} . This is done always in parallel, using the precomputed LU decompositions.