Effiency Issues of a Parallel FEM Implementation on Shared Memory Computers

Lothar Grabowsky
University of Technology Chemnitz–Zwickau
Dept. of Computer Science
D–09107 Chemnitz
grabowsk@informatik.tu-chemnitz.de

Wolfgang Rehm
University of Technology Chemnitz–Zwickau
Dept. of Computer Science
D–09107 Chemnitz
rehm@informatik.tu-chemnitz.de

Abstract

In the field of parallel FEM methods a number of highly efficient solutions for distributed memory systems are existing, but the passage to adaptive parallel FEM simulations leads, in all probability, to a more dynamic behaviour with respect to data placement and load balancing. Therefore shared–memory architecture seems to be a more appropriate solution for getting efficient implementations.

This work presents a parallelized CG–method for shared memory systems which was implemented on a 4–processor SMP system and makes explicit use of shared memory to enhance the communication between different domains. It is based on an idea for implementing parallization on distributed memory systems and represents an appropiate modification of this method.

The results show that an increased synchronization expense can partially compensate the advantages of shared memory communication depending on the levels of refinement and the processor number.

1. FEM substructure technique

In this Section we will give a short summary of the needed theoretical results. For more details see [2] and the references within.

We consider a symmetric, uniformly elliptic boundary value problem for a partial differential equation of second degree on a bounded domain $\Omega \subset \mathbb{R}^d$, $(d = 2, 3)$ with a piecewise smooth boundary Γ. The weak formulation of this problem leads to a symmetric, $\mathbb{V}_0$– elliptic and $\mathbb{V}_0$–bounded variation problem of the form:

$$\text{find } u \in \mathbb{V}_0 : \qquad a(u,v) = < F, v > \qquad \forall v \in \mathbb{V}_0 \quad (1)$$

We divide Ω into p non–overlapping subdomains, also called substructures or superelements, such that

$$\overline{\Omega} = \bigcup_{i=1}^{p} \overline{\Omega}_i \qquad \text{and} \qquad \Omega_i \bigcap \Omega_j = \emptyset \quad \text{for} \quad i \neq j$$

Let us divide the subdomains $\overline{\Omega}_i$ into finite elements $\overline{\delta}_r$, such that the discretisation process results in a conform triangulation of $\overline{\Omega}_i$. In the following the indices "C" and "I" denote quantities corresponding to the coupling boundary $\Gamma_C = \bigcup_{i=1}^{p} \partial\Omega_i \backslash \Gamma_D$ and to the interior of the subdomains $\Omega_1, \ldots, \Omega_p$, respectively, where Γ_D denotes the Dirichlet–boundary.

Let

$$\Phi = \{ \varphi_1, \ldots, \varphi_{N_C}, \varphi_{N_C+1}, \ldots, \varphi_{N_C+N_{I,1}}, \ldots, \varphi_{N=N_C+N_I} \} \quad (2)$$

the usual nodal basis, where the first N_C functions belong to nodes from coupling boundary Γ_C, the next $N_{I,1}$ functions to inner nodes from Ω_1, and so on. The FE–subspace

$$\mathbb{V} = \mathbb{V}_h \subset \mathbb{V}_0 \quad (3)$$

is now defined by the finite dimensional space:

$$\mathbb{V} = span(\Phi V) \quad (4)$$

with

$$V = (V_C\, V_I) = I = \begin{pmatrix} I_C & 0 \\ 0 & I_I \end{pmatrix}_{N \times N}$$

Once the basis Φ for $\mathbb{V}$ is chosen, the FE–approximation

$$\begin{gathered} \text{find } u = \Phi V \underline{u} \in \mathbb{V} : \\ a(\Phi V \underline{u}, \Phi V \underline{v}) = < F, \Phi V \underline{v} > \quad \forall v = \Phi V \underline{v} \in \mathbb{V} \end{gathered} \quad (5)$$

to (1) results in the system

$$K \underline{u} = \underline{f} \quad (6)$$

where K and f are defined by

$$\begin{aligned}(K\underline{u},\underline{v}) &= (V^T K V\underline{u},\underline{v}) \\ &= a(\Phi V\underline{u},\Phi V\underline{v}) \quad \forall \underline{u},\underline{v}\in\mathbb{R}^N\end{aligned} \tag{7}$$

$$(\underline{f},\underline{v}) = (\underline{f},V\underline{v}) =< F,\Phi V\underline{v} > \quad \forall \underline{v}\in\mathbb{R}^N \tag{8}$$

The f.e. isomorphism between $u\in\mathbb{V}$ and $\underline{u}=\left(\underline{u}_C^T\ \underline{u}_I^T\right)\in\mathbb{R}^N$ is given by

$$\mathbb{V}\ni u=\Phi V\underline{u}=\Phi\underline{u} \quad \overset{\Phi}{\longleftrightarrow} \quad \underline{u}\in\mathbb{R}^N \tag{9}$$

Taking into account the arrangement of the basis function given in (2) we can rewrite the system (6) in the block form

$$\begin{pmatrix} K_C & K_{CI} \\ K_{IC} & K_I \end{pmatrix}\begin{pmatrix} \underline{u}_C \\ \underline{u}_I \end{pmatrix} = \begin{pmatrix} \underline{f}_C \\ \underline{f}_I \end{pmatrix} \tag{10}$$

with

$$K_I = diag(K_{I,i})_{i=1,2,\ldots,p} \tag{11}$$

The well known FE substructure technique gives a factorization of K

$$K=\begin{pmatrix} I_C & K_{CI}K_I^{-1} \\ O & I_I \end{pmatrix}\begin{pmatrix} S_C & O \\ O & K_I \end{pmatrix}\begin{pmatrix} I_C & O \\ K_I^{-1}K_{IC} & I_I \end{pmatrix}$$

containing the Schur complement
$S_C = K_C - K_{CI}K_I^{-1}K_{IC}$.

In section 3 we will apply a via Domain Decomposition parallelized and preconditioned CG–method to system (10).

2. The hierarchical DD-preconditioning

In the message passing algorithm described in [2] a requirement is, that a preconditioner should not increase the communication amount significantly. Various preconditioners satisfying this condition were presented in [1, 2]. One example that was used for the practical experiments is described here. This method is equivalent to the hierarchical preconditioning by H. Yserentant (see [4, 5]). For this reason we give only the basic theoretical facts here, a detailed decription can be found in the publications above.

Starting point is a coarse grid (user triangulation). By this triangulation (consisting of as few triangles as possible) the geometry of the domain Ω should be sufficiently described. Additionally we assume that these triangles are used to define the subdomains Ω_i. Hence it is favourable if the number of triangles is a multiple of the number of processors. We call the nodes in this start triangulation nodes in level 0. Each node in level 0 is assigned a usual f.e. basis function $\psi_i = \varphi_{h_0,i}$, i.e. these functions form a nodal basis for the level 0 triangulation. Now the triangulation is refined hierarchically in l steps. Again we assign the usual basis functions $\psi_i = \varphi_{h_j,i}$ to nodes supervened in level j. At the end of this process we have defined N_0 basis functions in level 0, N_1 in level 1 an so on. The totality of all $N = N_0 + N_1 + \ldots + N_l$ basis functions is called hierarchical basis

$$\Psi = \{\psi_1,\ldots,\psi_{N_0},\psi_{N_0+1},\ldots,\psi_N\} \tag{12}$$

of the f.e. subspace $\mathbb{V} = span(\Psi) = span(\Phi)$. If we use this basis instead of the usual nodal basis, then the associated matrix $\hat{K} = (a(\Psi_i,\Psi_j))_{i,j=1,\ldots,N}$ would have the following properties: (see [3])

1. $\hat{K}$ is not a sparse matrix
2. The condition number $\kappa(\hat{K}) = O(|\ln h|^2)$

Because of 2. the conjugate gradient method (without preconditioning) would be a fast solver for sytems with the system matrix $\hat{K}$. On the other hand 1. leads to an increased memory expense to store $\hat{K}$ and a higher computational expense.

But if we have a basis transformation

$$\Psi = \Phi\hat{V} \tag{13}$$

with a regular $N\times N$-matrix $\hat{V}$ from (13) follows:

$$\hat{K} = \hat{V}^T K\hat{V} \tag{14}$$

and hence

$$\kappa(\hat{K}) = \kappa(\hat{V}^T K\hat{V}) = \kappa(\hat{V}\hat{V}^T K)$$

Therefore the matrix

$$C^{-1} = \hat{V}\hat{V}^T$$

is a good preconditioner for the System $K\underline{u} = \underline{f}$ in the nodal basis Φ.

The matrix multiplication

$$\underline{w} = \hat{V}\hat{V}^T\underline{r}$$

can be performed very efficiently (only N multiplications and 2N additions are needed).

3. A parallized CG–method for distributed memory systems

At first, we will introduce a parallel algorithm, developed for distributed memory computers (see also [2]). From this the required changes in the case of shared memory computers will be derived.

According to the p subdomains $\overline{\Omega}_i,\ (i = 1, 2, \ldots, p)$ the matrices and vectors are distributed over the processors. For the vectors we use two types of distribution called overlapping (type I) and adding (type II):

type I: $\underline{u}, \underline{w}, \underline{s}$ are stored in P_i ($\hat{=} \overline{\Omega}_i$)
as $\underline{u}_i = A_i\underline{u}, \underline{w}_i = A_i\underline{w}, \underline{s}_i = A_i\underline{s}$

type II: $\underline{r}, \underline{v}, \underline{f}$ are stored in P_i as
$\underline{r}_i, \underline{v}_i, \underline{f}_i$ such that $\underline{r} = \sum_{i=1}^{p} A_i^T \underline{r}_i$ etc.

where the $(N_i \times N)$-matrix A_i is the Boolean superlement connectivity matrix which maps overall vectors of nodal parameters onto the superelement vector of parameters associated with the subdomain $\overline{\Omega}_i$ only $(i = 1, \ldots, p)$. With these matrices K can be rewritten in the form:

$$K = \sum_{i=1}^{p} A_i^T K_i A_i \tag{15}$$

Using these data distribution and the notations introduced above, we can formulate the following parallized form of CG algorithm:

	for $i = 1, 2, \ldots p$ do in parallel
0.	Start Step Choose an initial guess $\underline{u}$, e.g. $\underline{u} = O$ $\underline{r}_i = \underline{f}_i - K_i\underline{u}_i$ $\underline{w} = C^{-1} \sum A_i^T \underline{r}_i$ $\underline{s}_i = \underline{w}_i \quad (\underline{w}_i = A_i\underline{w})$ $\sigma_i = \underline{w}_i^T \underline{r}_i$ $\sigma = \sigma^0 = \sum \sigma_i$
	Iteration
1.	$\widetilde{\underline{v}}_i = K_i\widetilde{\underline{s}}_i$ $\delta_i = \widetilde{\underline{v}}_i^T \widetilde{\underline{s}}_i$ $\delta = \sum \delta_i$ $\alpha = \widetilde{\sigma}/\delta$
2.	$\underline{u}_i = \widetilde{\underline{u}}_i + \alpha\widetilde{\underline{s}}_i$ $\underline{r}_i = \widetilde{\underline{r}}_i - \alpha\widetilde{\underline{v}}_i$
3.	$\underline{w} = C^{-1} \sum A_i^T \underline{r}_i$
4.	$\sigma_i = \underline{w}_i^T \underline{r}_i$ $\sigma = \sum \sigma_i$ $\beta = \sigma/\widetilde{\sigma}$
5.	$\underline{s}_i = \underline{w}_i + \beta\widetilde{\underline{s}}_i$
6.	$\sigma \leq \epsilon^2 \cdot \sigma^0$? $\xrightarrow{\text{no}}$ goto 1 $\downarrow$yes stop

In the algorithm above the tilde symbol "~" marks the old iterates. The sum sign $\sum$ means both summing up and transfer operations. Even in the case $C = I$ (no preconditioning) a data exchange of $O\left(h^{-(d-1)}\right)$ coupling boundary data is needed to perform step 3. So it is strongly required that the preconditioner C should not change this situation significantly.

3.1. The Preconditioning in the distributed memory case

The parallization of the multiplications

$$\underline{y} = \hat{V}^T \underline{r} \qquad \text{and} \qquad \underline{w} = \hat{V}\underline{y}$$

(see section 2) can be done in the following way. The vector $\underline{r}$ is stored as type II

$$\underline{r} = \sum_{i=1}^{p} A_i^T \underline{r}_i$$

The local multiplication

$$\underline{y}_i = \hat{V}_i^T \underline{r}_i$$

leaves the type II property of $\underline{y}$ unchanged, i.e. the equation

$$\underline{y} = \sum_{i=1}^{p} A_i^T \underline{y}_i$$

holds. If we generate a vector $\widetilde{\underline{y}}$ of type I

$$\widetilde{\underline{y}}_i = \underline{y}_i + \sum_{\substack{j=1 \\ j \neq i}}^{p} A_j^T \underline{y}_j$$

then the second multiplication

$$\underline{w}_i = \hat{V}_i^T \widetilde{\underline{y}}_i$$

again does not change the type I form of $\underline{w}$.

Hence a CG-step with hierarchical preconditioning leads to the same communication expense as without any preconditioning. For a more detailed description see [2].

4. A parallelized CG–method for shared memory systems

The aim of this implementation was to make explicit use of shared memory for parallelization. From this it seemed to be naturally to work with global datafields and not to split the matrices and vectors in local parts. The base to realize this was the multithreading programming model.

The domain decomposition underlies the parallelization as in the message passing case, but contrary to this the coupling nodes (which are not splitted in local parts) are used by several processors, what causes problems for the parallelization.

A consequence is that every thread uses a part of the global matrix K and not the superelementmatrices K_i. For all components assigned to inner nodes this obviously makes no difference. For nodes assigned to the coupling boundary a unique assignment to the threads needs to be found. For example the matrix K can be divided as follows

$$\left(\widetilde{K}_s\right)_{ij} = \begin{cases} k_{ij} & : \quad \omega_i, \omega_j \in \overline{\Omega}_s \\ & \quad \wedge \left(s = \min\limits_r \left\{ \omega_i \in \overline{\Omega}_r \right\} \vee \right. \\ & \quad \left. s = \min\limits_r \left\{ \omega_j \in \overline{\Omega}_r \right\} \right) \\ 0 & : \quad \text{else} \end{cases} \tag{16}$$

where $\wedge$ and $\vee$ denotes logical "and" and "or", respectively. The minimum condition can be replaced by any condition that guarantees an unique assignment of components to threads.

Then the equation

$$K = \sum_{i=1}^{p} \widetilde{K}_i \tag{17}$$

obviously holds.

The corresponding partial vectors can be defined by

$$(\underline{x}_s)_i = \begin{cases} x_i & : \quad \omega_i \in \overline{\Omega}_s \wedge s = \min\limits_r \left\{ \omega_i \in \overline{\Omega}_r \right\} \\ 0 & : \quad \text{else} \end{cases} \tag{18}$$

Additionally we define the following vectors

$$(\overline{\underline{x}}_s)_i = \begin{cases} x_i & : \quad \omega_i \in \overline{\Omega}_s \\ 0 & : \quad \text{else} \end{cases} \tag{19}$$

Remark

Since all threads have access to the whole matrix K other divisions of K are possible too. So a distribution by rows can be accomplished. This guaranties, that every vector component is changed by only one thread and therefore these operations can be performed unprotected. On the other hand this leads to a loss of locality with respect to the data distribution in the assembly phase. The following considerations on the synchronization expense can be done analogously in this case.

With the notations above we can formulate the parallel algorithm:

for $i = 1, 2, \ldots p$ do in parallel	
0.	Start Step
	Choose an initial guess $\underline{u}$, e.g. $\underline{u} = O$
	$\underline{r}_i = \underline{f}_i$
	$\overline{\underline{r}}_i = \overline{\underline{r}}_i - \widetilde{K}_i \overline{\underline{u}}_i$
	$\underline{w} = \sum C^{-1} \underline{r}_i$
	$\underline{s}_i = \underline{w}_i$
	$\sigma_i = \underline{w}_i^T \underline{r}_i$
	$\sigma = \sigma^0 = \sum \sigma_i$
	Iteration
1.	$\widetilde{\underline{v}}_i = 0$
	$\widetilde{\overline{\underline{v}}}_i = \widetilde{\overline{\underline{v}}}_i + \widetilde{K}_i \widetilde{\overline{\underline{s}}}$
	$\delta_i = \widetilde{\underline{v}}_i^T \widetilde{\underline{s}}_i$
	$\delta = \sum \delta_i$
	$\alpha = \widetilde{\sigma} / \delta$
2.	$\underline{u}_i = \widetilde{\underline{u}}_i + \alpha \widetilde{\underline{s}}_i$
	$\underline{r}_i = \widetilde{\underline{r}}_i - \alpha \widetilde{\underline{v}}_i$
3.	$\underline{w} = \sum C^{-1} \underline{r}_i$
4.	$\sigma_i = \underline{w}_i^T \underline{r}_i$
	$\sigma = \sum \sigma_i$
	$\beta = \sigma / \widetilde{\sigma}$
5.	$\underline{s}_i = \underline{w}_i + \beta \widetilde{\underline{s}}_i$
6.	$\sigma \leq \epsilon^2 \cdot \sigma^0$? $\xrightarrow{\text{no}}$ goto 1
	$\downarrow$yes
	stop

In the algorithm above the tilde symbol "~" marks the old iterates.

After steps 1 and 4 a global synchronization is necessary in order to make sure that the scalar products are completely evaluated. Additionally you need a synchronization between step 5 an step 1. Here threads that have common nodes have to be synchronized, and this synchronization is really additional compared to the message passing algorithm described in section 3. Step 3 will be discussed in the next Section. We only note here that in the case $C = I$ this step results in the local operation $\underline{w}_i = \underline{r}_i$.

4.1. The Preconditioning in the shared memory case

We have to perform the multiplications

$$\underline{y} = \hat{V}^T \underline{r} \qquad \text{and} \qquad \underline{w} = \hat{V} \underline{y}$$

with V from section 2. This is done by a factorisation

$$\hat{V} = \hat{V}^{(l)} \hat{V}^{(l-1)} \ldots \hat{V}^{(1)} \tag{20}$$

(see [4]). Now we define matrices analogous to $\widetilde{K}_i$:

$$\left(\hat{V}_s^{(k)}\right)_{ij} = \begin{cases} v_{ij}^{(k)} & : \quad \omega_i, \omega_j \in \overline{\Omega}_s \\ & \quad \wedge \left(s = \min\limits_r \left\{\omega_i \in \overline{\Omega}_r\right\} \vee \right. \\ & \quad \left. s = \min\limits_r \left\{\omega_j \in \overline{\Omega}_r\right\}\right) \\ 0 & : \quad \text{else} \end{cases} \tag{21}$$

$$k = 1, 2, \ldots, l$$

where

$$v_{ij}^{(k)} = \left(\hat{V}^{(k)}\right)_{ij}$$

Then step 3 of the parallel algorithm described in Section 4 leads to

$$\begin{aligned} &\underline{y}_i = 0, \\ &\overline{\underline{y}}_i = \overline{\underline{y}}_i + \left(\hat{V}_i^{(1)}\right)^T \ldots \left(\hat{V}_i^{(l-1)}\right)^T \left(\hat{V}_i^{(l)}\right)^T \underline{r}_i \end{aligned} \tag{22}$$

$$\underline{w}_i = \hat{V}_i^{(l)} \hat{V}_i^{(l-1)} \ldots \hat{V}_i^{(1)} \overline{\underline{y}}_i \tag{23}$$

If we rewrite (22) in the form

$$\begin{aligned} &\underline{y}_i^{(1)} = 0, && \overline{\underline{y}}_i^{(1)} = \overline{\underline{y}}_i^{(1)} + \left(\hat{V}_i^{(l)}\right)^T \underline{r}_i \\ &\underline{y}_i^{(k+1)} = \underline{y}_i^{(k)}, && \overline{\underline{y}}_i^{(k+1)} = \underline{y}_i^{(k+1)} + \left(\hat{V}_i^{(l-k)}\right)^T \underline{y}_i^{(k)} \\ & && (k = 1, \ldots, l-1) \\ &\underline{y}_i = \underline{y}_i^{(l)} \end{aligned}$$

and (23) as follows

$$\begin{aligned} &\underline{w}_i^{(1)} = \hat{V}_i^{(1)} \overline{\underline{y}}_i \\ &\underline{w}_i^{(k+1)} = \hat{V}_i^{(k+1)} \overline{\underline{w}}_i^{(k)} \quad (k = 1, \ldots, l-1) \\ &\underline{w}_i = \underline{w}_i^{(l)} \end{aligned}$$

it is obvious that a synchronization is needed after each level, i.e. $(2 \cdot l)$ synchronizations are needed to perform the preconditioning.

4.2. Results

In order to verify that you can increase the efficiency by using multithreading on shared memory systems we have implemented two versions of an example program, a message passing (according to the algorithm described in section 3) and a multithreading version. These program versions were evaluated on a pentium based system with 4 processors (Compaq Proliant 4000) and on a KSR1 system with 8 processors.

As described in 4.1 the synchronization expense increases with the number of levels, whereas the communication expense in the message passing program is independent from the number of levels. Thus it is to expect, that the advance of efficiency in the multithread program will be decreased if the number of levels is increased.

The results shown in fig. 1 confirm this.

2 processors

4 processors

Figure 1. speedup results on the KSR1

5. Conclusions

The results show that the explicit use of shared memory based on multithreading techniques can lead to an increased efficiency on shared memory systems. On the other hand an increased synchronization expense can partial compensate this. Moreover the expense to transpose the algorithm to

multithreading is relatively high. With more difficult preconditioners and in the 3d-case this expense will be additionally increased (due to more complex data dependencies). From this reasons we are concerned with the question how to use multithreading inside the message passing library. From this we expect similar increased efficiency but without expensive changes in the application.

References

[1] G. Haase, U. Langer, and A. Meyer. A new approach to the Dirichlet domain decomposition method. In S. Hengst, editor, *Fifth Multigrid Seminar, Eberswalde, 1990*, pages 1–59. Karl–Weierstrass–Institut, 1990. Report R–MATH–09/90.

[2] G. Haase, U. Langer, and A. Meyer. Parallelisierung und Vorkonditionierung des CG-Verfahrens durch Gebietszerlegung. In *Proceedings of the GAMM-Seminar "Numerische Algorithmen auf Transputersystemen" held at Heidelberg, 1991*. Teubner Verlag, Stuttgart, 1991.

[3] A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexat solvers on each subdomain. *Computing*, 45:217–234, 1990.

[4] H. Yserentant. On the multi–level splitting of finite element spaces. *Numer. Math.*, 49(4):379–412, 1986.

[5] H. Yserentant. Two preconditioners based on the multi–level splitting of finite element spaces. *Numer. Math.*, 58:163–184, 1990.