Technische Universität Chemnitz
Sonderforschungsbereich 393
Numerische Simulation auf massiv parallelen Rechnern

Mathias Meisel Arnd Meyer

Hierarchically Preconditioned Parallel CG–Solvers with and without Coarse–Mesh–Solvers inside FEAP

Preprint SFB393/97-20

Fakultät für Mathematik
TU Chemnitz-Zwickau
D-09107 Chemnitz, FRG
(0371)-531-2657 (fax)

mathias.meisel@mathematik.tu-chemnitz.de
http://www.tu-chemnitz.de/~mmeisel
(0371)-531-4105

arnd.meyer@mathematik.tu-chemnitz.de
http://www.tu-chemnitz.de/~amey
(0371)-531-2659

Preprint-Reihe des Chemnitzer SFB 393

SFB393/97-20 **September 1997**

Hierarchically Preconditioned Parallel CG–Solvers with and without Coarse–Mesh–Solvers inside FEAP

Mathias Meisel Arnd Meyer

September 30, 1997

Abstract

After some remarks on the parallel implementation of the Finite Element package FEAP, our realisation of the parallel CG–algorithm is sketched. From a technical point of view, a hierarchical preconditioner with and without additional global crosspoint preconditioning is presented. The numerical properties of this preconditioners are discussed and compared to a Schur–complement–preconditioning, using a wide range of data from computations on technical and academic examples from elasticity.

Contents

1 The parallelized FEAP–Version 1

2 The parallel conjugate gradients algorithm 2

3 Hierarchical preconditioning 3
3.1 The hierarchical list 3
3.2 The hierarchical preconditioner 4

4 Coars grid preconditioning 5
4.1 The coarse grid matrix 6
4.2 The crosspoint solver 7

5 Different technologies for the global assembly 7

6 Comparison to the Schur complement CG 8

7 Numerical results 9
7.1 The model problems 10
7.2 Grid geometry and preconditioning 10
7.3 Comparison of the preconditioners 13
7.4 Different crosspoint matrices 22

1 The parallelized FEAP–Version

In the last three years, together with the authors of [5], a lot of work was done to portage the sequential **F**inite **E**lement **An**alysis **P**ackage FEAP ([6]) to parallel computers with message passing architecture. We mainly used the operating system PARIX, but with small changes in some basic subroutines for communication and initialisation, the switch to other operating systems (PVM,...) is easy to manage. In the resulting parallel version of FEAP (||-FEAP), consisting of more than 40·000 lines of source code, the utilities for grafical post processing and interactive control of the computational algorithms are well combined with the distributed execution of those subtasks, which are especially intensive in computational effort. So, complete analysing computations to a wide class of linear or nonlinear problems from elasticity can be performed on parallel computers in a comfortable and efficient manner.
As the principle of parallelisation we used the concept of domain decomposition. The input data, describing the problem to solve, is formulated in some kind of meta language, read in from a file by one distinguished processor and distributed to all others using a tree like topology of the processors. Conversly, the output of results, including X11 or Postscript based graphics, is completely prepared in parallel and sequentialised by a ring topology. The mesh generation and the assembly of the local[1] stiffnes matrices are performed in parallel without any communication, because each processor can operate autonomously on his subdomain.
Since large but sparse linear systems are typical for those finite element applications, we prefer preconditioned iterative solvers. The first solver implemented in ||-FEAP was a conjugate gradient algorithm combined with an exact solver used to compute the preconditioned residuals on the inner nodes of the subdomains and a Laplace like preconditioner for the Schur complement at their boundaries. This solver and a lot of data on the number of iterations and the computational time necessary to solve various problems on different meshes and varying numbers of processors as well as on speed up and efficiency are described in detail in [2].
In each step of the iteration procedure one global assembly of the residual vector is needed. Therefore, the efficiency of the whole solver especially depends on the technology of communication and data exchange between the processors. In our first implementation, the global assembly was based on a hypercube topology of the involved processors. Developing two different topologies, the efficiency of this task could be highly improved by explicitly utilizing the geometrical relations between the subdomains. This is documented in [3].
The aim of this work is to present a new implemented hierarchical preconditioner based on the ideas of Yserentant [7] and the experiences we made with it. An additional preconditioning for the crosspoint system will also be described.
Since the method of conjugate gradients to solve a system $\mathbf{K}\mathbf{x} = \mathbf{b}$ of linear equations is quite common, the algorithm implemented in ||-FEAP shall only be sketched at this place.

[1] with respect to a single subdomain

2 The parallel conjugate gradients algorithm

Let $\mathbf{C}$ be some preconditioner for $\mathbf{K}$, $\mathbf{x}^0$ an initial guess for the solution $\mathbf{x}$, $i:=0$, $\alpha^0:=1$, $\beta^0:=0$ and $\mathbf{u}^0:=\mathbf{0}$. Then, our algorithm starts up with

$$\begin{array}{rcll}
\mathbf{r}^0 & := & \mathbf{K}\mathbf{x}^0-\mathbf{b} & \\
\mathbf{w}^0 & := & \mathbf{C}^{-1}\mathbf{r}^0 & (\star) \\
\mathbf{h}^0 & := & \mathbf{K}\mathbf{w}^0 & \\
\mathbf{q}^0 & := & \mathbf{w}^0 & \\
\gamma^0 & := & (\mathbf{w}^0,\mathbf{r}^0) & (\bullet) \\
\tau^0 & := & (\mathbf{w}^0,\mathbf{h}^0) & (\bullet)
\end{array}$$

and runs through

$$\begin{array}{lrcll}
1: & \mathbf{u}^{i+1} & := & \mathbf{h}^i+\beta^i\mathbf{u}^i & \\
 & \alpha^{i+1} & := & -\left[\frac{\tau^i}{\gamma^i}+\frac{\beta^i}{\alpha^i}\right]^{-1} & \\
 & \mathbf{x}^{i+1} & := & \mathbf{x}^i+\alpha^{i+1}\mathbf{q}^i & \textbf{STOP} \text{ if } \gamma^i \text{ is small enough} \\
 & \mathbf{r}^{i+1} & := & \mathbf{r}^i+\alpha^{i+1}\mathbf{u}^i & \\
 & \mathbf{w}^{i+1} & := & \mathbf{C}^{-1}\mathbf{r}^{i+1} & (\star) \\
 & \mathbf{h}^{i+1} & := & \mathbf{K}\mathbf{w}^{i+1} & \\
 & \gamma^{i+1} & := & (\mathbf{w}^{i+1},\mathbf{r}^{i+1}) & (\bullet) \\
 & \tau^{i+1} & := & (\mathbf{w}^{i+1},\mathbf{h}^{i+1}) & (\bullet) \\
 & \beta^{i+1} & := & \gamma^{i+1}/\gamma^i & \\
 & \mathbf{q}^{i+1} & := & \mathbf{w}^{i+1}+\beta^{i+1}\mathbf{q}^i & \\
 & i & := & i+1 & \textbf{GO TO } 1
\end{array} \tag{2.1}$$

In the nodes on the <u>coupling boundaries</u> and in the <u>crosspoints</u>[2] of the subdomains there are two possibilities to distribute values of matrix entries and vektor components to the concerned processors. The first is that each processor owning the regarded node posseses a copy of those values. This is used for the vectors $\mathbf{w}$, $\mathbf{q}$ and $\mathbf{x}$ in (2.1). In the second, used for $\mathbf{r}$, $\mathbf{u}$, $\mathbf{h}$ and $\mathbf{b}$ as well as for the entries of $\mathbf{K}$, the true value may be obtained by summing up the contributions of all processors owning the regarded node. The transformation from the second to the first kind ("global assembly") requires communication between the processors. In our CG procedure, this happens only in the steps marked with a "$\star$ ". The <u>expense</u>[3] of this communication depends on the used communication technology, cf.[3], and grows with decreasing mesh size h and with P, the number of processors. To obtain the global scalar product in the steps marked with a "$\bullet$", the local parts must be collected over all processors with an expense not depending on h and only growing with P. This operation can be done in one step for both numbers γ and τ. In [2] and summarizing in [3] it is shown, that all other operations of (2.1) can be performed in parallel without any additional communication requirements.

[2]edges common to 2 subdomains and vertices, belonging to at least 2 subdomains

[3]number of communication actions and amount of exchanged data per iteration step

The main disadvantages of using an exact solver to compute the preconditioned residuum $\mathbf{w}$ at the inner nodes of the subdomains are the large amount of storage needed for the decomposed submatrices of $\mathbf{K}$ and the expense of arithmetic to compute these factorization and the inner residuals.
To overcome this, the hierarchical preconditioner presented in the following section was implemented.

3 Hierarchical preconditioning

Neither the original version of FEAP nor the recent version of ||-FEAP are designed to generate hierarchical meshes. Therefore, for meshes that could have been generated by some hierarchical mesh generator, an algorithm for computing a hierarchical list to such an existing mesh was implemented.

3.1 The hierarchical list

Suppose, the considered domain Ω is decomposed into P quadrilateral subdomains Ω_s $(0 \leq s \leq P-1)$ and these subdomains are distributed to P processors. Assume, each processor $\mathcal{P}_s$ has generated a quadrilateral mesh on Ω_s, consisting of $(2^{N_s^x}+1) \times (2^{N_s^y}+1)$ grid points. The totality of all these submeshes defines a FE–grid over Ω without "hanging nodes". Then, ||-FEAP is able to compute the hierarchical list for each subdomain in parallel and without communication, and their totality is a hierarchical list of the whole mesh.

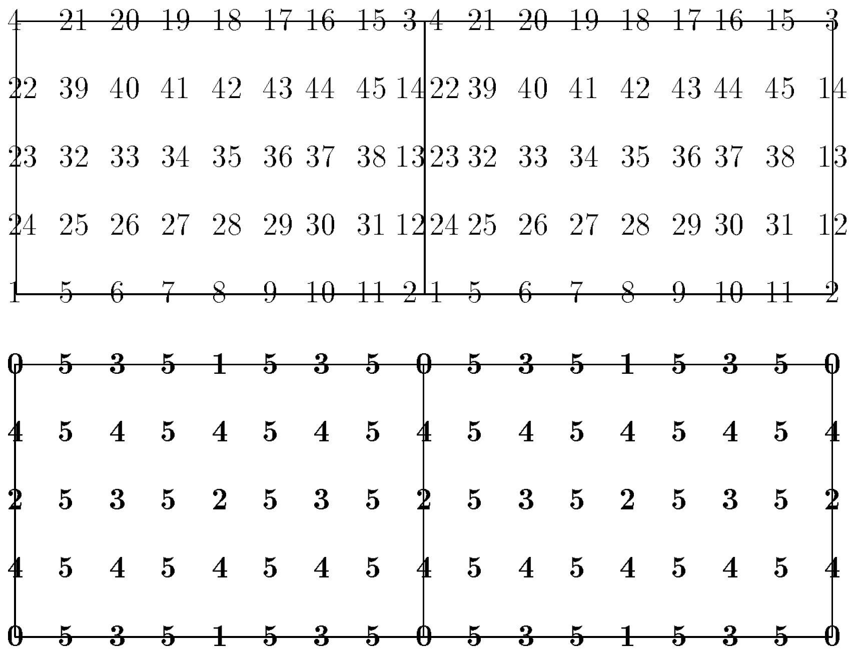


Figure 1: Example of a hierarchicalised mesh in 2 subdomains

In figure 1, the local numbering of the nodes used inside ||-FEAP (above) and the membership of the nodes to the hierarchical levels (below) are displayed for the case of 2 subdomains with $N_0^x = N_1^x = 3$ and $N_0^y = N_1^y = 2$.

The main idea to get the hierarchical lists is the following:
Let the boundary and the vertices of the subdomain (level 0) be fixed. Then, alternating in horizontal and vertical direction[4], new lines are inserted, defining a new level, and the points of intersection with the just existing lines are added to the hierarchical list at this new level als "sons" of the two nodes (fathers) in the neighbourhood on the intersected line.
For each subdomain from figure 1 this leads to the list presented in table 1.

son	father 1	father2		son	father 1	father2		son	father 1	father2
	level 1				level 4			5	1	6
8	1	2		22	4	23		19	18	20
18	3	4		40	20	33		41	40	42
	level 2			42	18	35		34	33	35
23	1	4		44	16	37		27	26	28
35	8	18		14	3	13		7	6	8
13	2	3		24	1	23		17	16	18
	level 3			26	6	33		43	42	44
20	4	18		28	8	35		36	35	37
33	23	35		30	10	37		29	28	30
6	1	8		12	2	13		9	8	10
16	3	18			level 5			15	3	16
37	13	35		21	4	20		45	14	44
10	2	8		39	22	40		38	13	37
				32	23	33		31	12	30
				25	24	26		11	2	10

Table 1: Hierarchical list to figure 1

3.2 The hierarchical preconditioner

In elasticity problems, there are typically $I > 1$ degrees of freedom associated to each single node in the computational grid. They shall be numbered by i , $1 \leq i \leq I$, and the total number of unknowns is $N := I \cdot \sum\limits_{s=0}^{p-1} (2^{N_s^x} + 1)(2^{N_s^y} + 1)$.
Except the four vertices of the subdomain (level 0), each node in the hierarchical(ized) mesh is the "son" of exactly two "fathers", cf. table 1, and for a mesh consisting of $(2^{N_s^x} + 1) \times (2^{N_s^y} + 1)$ grid points the hierarchical list consists of $M_s := (2^{N_s^x} + 1)(2^{N_s^y} + 1) - 4$ lines. Let these lines be numbered, starting from the lowest level 1 and ending at the highest level $N_s^x + N_s^y$, with m , $1 \leq m \leq M_s$. With thist conventions, the notation S_m, F_m^1 and F_m^2 for the "son" and the two "fathers" in the m'th row of the hierarchical list is used to define the linear operator

$$\mathbf{Q}_s \ : \qquad v_{S_m}^i := v_{S_m}^i + \frac{1}{2}\left(v_{F_m^1}^i + v_{F_m^2}^i\right) , \qquad \begin{matrix} i = 1\ (1)\ I \\ m = 1\ (1)\ M_s \end{matrix} \tag{3.1}$$

[4]in case of $|N_s^x - N_s^y| > 1$ in a more flexible way to distribute the directions as evemly as possible

acting on some grid function $\mathbf{v}$, and it's transposed

$$\mathbf{Q}_s^T \; : \qquad v^i_{F^j_m} := v^i_{F^j_m} + \frac{1}{2} v^i_{S_m} \, , \qquad \begin{matrix} j=1,2 \\ i=1\,(1)\,I \\ m=M_s\,(-1)\,1 \end{matrix} \quad . \tag{3.2}$$

In (3.1), the half of the sum of the values of both "fathers" is added to the value of the "son", whereas in (3.2) the half of the value of the "son" is added to the values of both "fathers".
As the collectivity of the local hierarchical lists defines the hierarchical list over the whole domain, the local matrices $\mathbf{Q}_s$ and $\mathbf{Q}_s^T$ define the two parts $\mathbf{Q}$ and $\mathbf{Q}^T$ of the hierarchical preconditioner over the whole grid.
Since nodes with essential boundary conditions cannot be removed from the grid and the preconditioned residual $\mathbf{w}^i$ has to be zero for all times whenever the initial solution fulfills this conditions, we need the diagonal matrix Ψ of dimension N

$$\Psi := \, diag\,(\Psi_{ll}) := \begin{cases} 0 & essential\ conditions \\ 1 & all\ others \end{cases} \tag{3.3}$$

to cut off pollutions. The matrix

$$\mathbf{J} := \Psi \; diag\,(k_{ll}^{-1}) \tag{3.4}$$

of the same dimension and derived from the stiffness matrix $\mathbf{K}$ is needed to describe an additional Jacobi scaling. Finally let Σ denote the operation of the global assembly of a vector (cf. section 5).
Then, the hierarchical preconditioner $\mathbf{C}$ in (2.1) may be expressed by[5]

$$\mathbf{C}^{-1} := \Psi \; \mathbf{Q} \; \Sigma \; \mathbf{J} \; \mathbf{Q}^T \Psi \quad . \tag{3.5}$$

Except the assembly Σ, all operations in (3.5) can be performed in parallel and without communication.
Applying Ψ or $\mathbf{J}$ to a vector is realized with the multiplication in components of two local vectors using standard routines described in [1], and $\mathbf{Q}$ as well as $\mathbf{Q}^T$ contain only simple operations acting on the same local vector.

4 Coars grid preconditioning

Regarding the loaded beam from figure 3, divided into 128 subdomains and distributet to 128 processors, we state, that only the subdomains Ω_s , $96 \leq s \leq 127$, are subject to the loading and no other than Ω_0 contains essential boundary conditions. Since, in each CG–step, exchange of information takes place only across the coupling boundaries between geometrically neighbouring subdomains, it takes

[5]The rightmost Ψ can be dropped, if $\mathbf{K}$ contains no other than diagonal entries in the rows associated to essential boundary conditions.

at least 96 steps until processor $\mathcal{P}_0$ gets some knowledge about the existence of the loading, and additional 127 steps are necessary to bring the first reaction forces from Ω_o to $\mathcal{P}_{127}$'s attention.
This heuristic considerations explain, why computations to the beam problem require, by the same number of degrees of freedom, essentially more iterations than the geometrically more compact problem from figure 2, when a large number of precessors is used[6].
The main idea of coarse grid preconditioning consists in regarding the domain decomposition as a computational grid and solving a suitable boundary value problem (BVP) on this grid to obtain improved residuals in the crosspoints. This BVP can be the original problem or a substitute with similar properties.

4.1 The coarse grid matrix

In case of quadrilateral subdomains, the most simple but in common use variant of constructing a coarse grid matrix is to regard the quadrilaterals as unit squares[7] and assemble the descrete Laplacian over this uniform grid. The resulting matrix shall be refered to by $\widetilde{\mathbf{L}}_S$. This requires no local computations and only a minimum of storage and communication, but the more the real geometry of the Ω_s differs from squares, the less is the coincidence to the original problem (cf. sections 7.4 and 7.2).
To overcome this, the Matrix $\widetilde{\mathbf{L}}_Q$, arising from the global assembly of the descrete Laplacian over the real quadrilateral grid, takes presedence. The storage and communication requirements are the same as for $\widetilde{\mathbf{L}}_S$ and so are the dimension and the bandwidth of the matrices. Because the geometrical computations must be performed only once bevor the iteration starts, the local computational effort can be left out of account.
Since the discrete Laplacians $\widetilde{\mathbf{L}}_S$ or $\widetilde{\mathbf{L}}_Q$ will be used separately for each of the I degrees of freedom associated to a single node in the grid, the coarse grid matrices $\mathbf{L}_S$ and $\mathbf{L}_Q$ are defined to be the block diagonal matrices consisting of I $\widetilde{\mathbf{L}}_S$ – or $\widetilde{\mathbf{L}}_Q$ –blocks, respectively.
The most expensive variant is to discretize the original problem on the coarse grid. In case of $I > 1$ degrees of freedom per single node, the dimenson of the resulting matrices $\mathbf{E}_S$ or $\mathbf{E}_Q$ is the same as the dimension of $\mathbf{L}_S$, but they are less sparse, their bandwith is larger and they do not decompose into I submatrices independent from each other.
Since a direct solver shall be used to solve the crosspoint problem, the matrix $\mathbf{L}_Q$ seems to be a good compromise.
In ||-FEAP, $\widetilde{\mathbf{L}}_Q$ is gained by discretising the Laplace operator with bilinear functions over all Ω_s and performing a global assembly on each $\mathcal{P}_s$ (cf. section 5). After that, each processor $\mathcal{P}_s$ performs a Cholesky decomposition with its own copy of $\widetilde{\mathbf{L}}_Q$,

[6]With the preconditioner (3.5) inside the algorithm (2.1) on 128 processors, 837 iterations were needed to solve the beam problem with 143550 degrees of freedom (dof), whereas the wedge problem with 143088 dof was solved after 136 steps.
[7]e.g., neglecting the real geometry

implicitly defining a decomposition for $\mathbf{L}_Q$:

$$\widetilde{\mathbf{L}}_Q = \widetilde{\mathbf{L}}\,\widetilde{\mathbf{L}}^T \qquad ; \qquad \mathbf{L}_Q = \mathbf{L}\,\mathbf{L}^T \quad . \tag{4.1}$$

4.2 The crosspoint solver

With the coarse grid matrix $\mathbf{L}_Q$ (or $\mathbf{L}_S$) from the previous section, $\mathbf{I}$ being the unit matrix of dimension $N - N^C$, N^C beeing the dimension of $\mathbf{L}$,

$$\mathbf{G}^{-1} := \begin{pmatrix} \mathbf{L}_Q^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} = \begin{pmatrix} (\mathbf{L}^T)^{-1}\mathbf{L}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \tag{4.2}$$

and using $\mathbf{Q}$, $\mathbf{Q}^T$, Σ, Ψ and $\mathbf{J}$ from (3.1)–(3.4), the hierarchical preconditioner with imbedded crosspoint preconditioning reads[8]

$$\mathbf{C}^{-1} := \Psi\;\mathbf{Q}\;\mathbf{J}^{1/2}\;\mathbf{G}^{-1}\Sigma\;\mathbf{J}^{1/2}\;\mathbf{Q}^T\;\Psi \qquad . \tag{4.3}$$

As in (3.5), the simple operations denoted by Ψ, $\mathbf{Q}^T$, $\mathbf{J}^{1/2}$ and $\mathbf{Q}$ work in parallel and require no communication. Compared to (3.5), some additional communicational effort araises from the crosspoint solver, but this shall be diskussed in section 5.
The root $\mathbf{J}^{1/2}$ should be precomputed before the iteration cycle starts. So, in comparison to (3.5), the additional numerical effort per iteration step caused by the crosspoint preconditioning is one diagonal scaling ($\mathbf{J}^{1/2}$) and I times the backward and the forward substitution associated to $(\widetilde{\mathbf{L}}^T)^{-1}\widetilde{\mathbf{L}}^{-1}$.That's the reason why the preconditioning (4.3) actually requires less steps of iteration than (3.5) but sometimes needs more computational time, when only a few steps are saved.

5 Different technologies for the global assembly

In [3], three different techniques for the assembly of the residuals in each CG–step were discussed in detail.
On the first and simpliest, basing on the routine "cube–cat" from [1] and referred to as "hypercube communication", each processor collects the data from all coupling nodes[9] of all other processors, seeks in the resulting storage vector[10] for components related to nodes owned by himself, too, and ignores the rest. Obviously, especially when a large number of processors is used, each processor uses only a small part of the interchanged data for the assembly of his residual values.
To reduce this overhead, the "coupling–edges communication" was developed. On this method, hypercube communication takes place only for the crosspoints, whereas the data along the interior of each coupling edge is exchanged directly between the two processors, whose subdomains share this edge. This requires the generation of

[8]footnote 5 from page 5 also applies to 4.3
[9]crosspoints and nodes at the coupling boundaries
[10]which might be huge in case of a small meshsize and/or a large number of processors

a virtual topology of the processors, reflecting the geometrical interrelationship of the subdomains.
For domain decompositions consisting of quadrilaterals only, this topology consists of up to 4 additional virtual links per processor.
If the quadrilateral subdomains define a tensor product mesh, a third technology named "crosspoint communication", uses no hypercube communication at all. Including the crosspoint data at the end of the coupling edges into the direct data exchange between the concerned processors, it remains to establish at each processor again up to 4 additional virtual links to those processors, whose subdomains intersect with the regarded subdomain in exactly one crosspoint, to exchange the crosspoint data with all processors needing them. The larger the number of used processors, the larger is the gain in time and storage.
If no coarse grid preconditioning is intended ($\mathbf{C}^{-1}$ from (3.5)), crosspoint communication is the most efficient technology, because of the fewest storage requirements and the least amount of transported data, but if $\mathbf{C}^{-1}$ shall be taken from (4.3), at least one processor must collect all crosspoint data to solve the coarse grid system. There are two possibilities in common use:

1. Only one processor assembles the coarse grid matrix, collects all coarse grid residuals, solves the crosspopint system and distributes the solution to all other processors. This works good with the routines like "tree–up" and "tree–down" from [1].

2. Each processor assembles his own coarse grid matrix, receives all coarse grid residuals, solves the crosspopint system and uses only those components of the resulting vector, assigned to the edges of his own patch. In this case, "cube–cat" should be used for the crosspoint data.

In ||-FEAP, the following is implemented: If the preconditioner (3.5) is used, the assembly of the residuals is based on "crosspoint communication", and in case of (4.3), the "coupling–edges communication" is choosen to meet the needs of the crosspoint solver in variant 2. Note, that an additional crosspoint solver causes not only additional numerical work per iteration step but also additional effort in communication.

6 Comparison to the Schur complement CG

The Schur complement CG algorithm (**SCCG**) is completly documented in [3], page 5, or in [2].
The essential computational expense per iteration consists of four linear combinations of vectors ($y+tz$), one matrix–vector–multiplication (Ky), two scalar products ((x,y)) and the work to be done in the preconditioner ($w=C^{-1}r$).
To compare both algorithms from a local point of view, let N_s^I and N_s^B denote the number of degrees of freedom located in the interior and at the boundary of the subdomain Ω_s.

In [2] for the **SCCG** it was shown that, due to the exact solver used as preconditioner in the interior of the subdomains, all the four operations $y + tz$ and the two scalar products can be restricted to the n_s^B components associated to the coupling boundaries, saving $12N_s^I$ FLOP's in each step of the algorithm. For the same reason, instead of the whole matrix–vector–multiplication $K_s w_s$ needed in the hierarchical case, only the submatrices K_s^B, K_s^{IB} and K_s^{BI} from $K_s = \begin{pmatrix} K_s^B & K_s^{BI} \\ K_s^{IB} & K_s^I \end{pmatrix}$ must be applied, saving at least additional $10N_s^I$ FLOP's. On the other hand, the number of nonzero entries in the Cholesky factorization of K_s^I used in the **SCCG** grows quadratically with N_s^I, overcompensating the saved $22N_s^I$ FLOP's for sufficiently large problems.

7 Numerical results

To compare the efficiency of the various preconditioners, the following three examples will be regarded. They are choosen to be at least a little bit realistic and, coincidental, make visible the expected benefits and disadvantages of the compared algorithms.

The following abbreviations will be used in this chapter:

- T_a and T_t are denoting the time needed for the arithmetic operations and the total time (including the time for communication).
- T^s, T^h and T^{hc} are denoting the time in seconds, needed to solve the problem with **SCCG**, with the hierarchical preconditioner (3.5) and with the hierarchical preconditioner with additional crosspointsolver (4.3).
- $\#^s$, $\#^h$ and $\#^{hc}$ are denoting the corresponding number of performed iterations to achieve the desired accuracy ($\gamma^i \leq 10^{-12}\gamma^0$ in (2.1)).
- $\#_S^{hc}$ indicates, that the coarse grid matrix was based on $\widetilde{\mathbf{L}}_S$ (see chapter 4.1).
- DIM denotes the total size of the problem, counting unknowns located at coupling nodes only once and reduced by the number of essential boundary conditions.
- $\mathcal{P}$ denotes the number of used processors.

All computations were performed on the "GC–PowerPlus"–computer under the PARIX environment.

7.1 The model problems

Wedge under surface loading
A wedge, fixed on the upper half of both slanting sides, is pressed from above by a piston (figure 2).

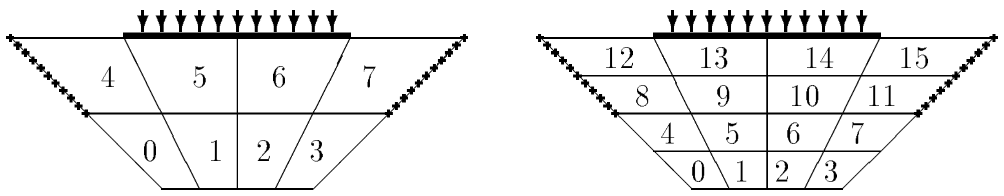


Figure 2: Wedge on 8 and on 16 processors

Beam under surface loading
A beam, fixed at one frontage, is loaded at the opposite end (figure 3).

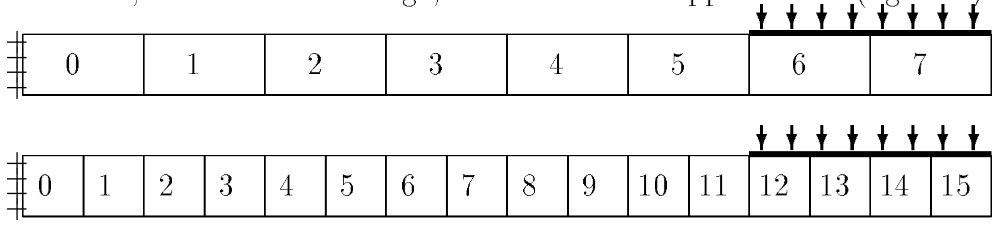


Figure 3: Beam on 8 and on 16 processors

Cooks membrane problem
As a classical technical problem we selected tho following (figure 4):

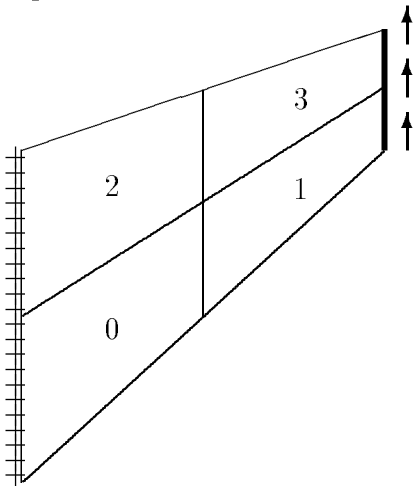
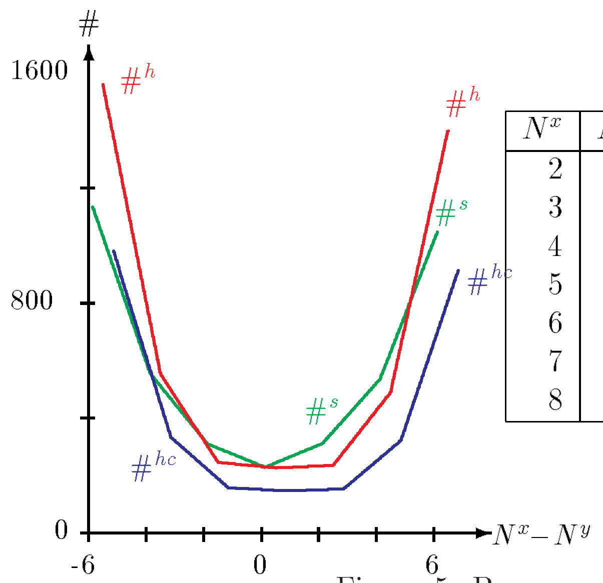


Figure 4: Cooks membrane problem on 4 processors

7.2 Grid geometry and preconditioning

The condition number of FE–matrices depends on the ratio ν of the smallest to the largest edge in the computational grid. To eleminate them from the other investigations, this influences shall be studied first. Since the N_s^x and N_s^y were choosen uniform for all s, the subscript s will be suppressed.

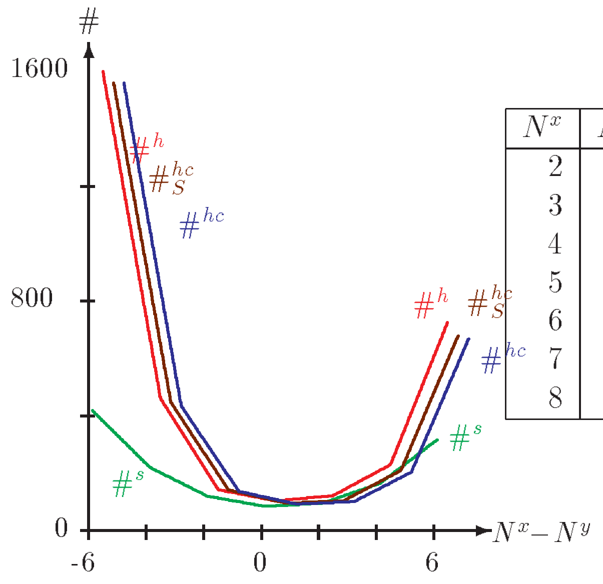
Since the subdomains in the beam problem from figure 3 on 16 processors are squares, we have the best mesh for $N^x = N^y$ and the more N^x differs from N^x, the worse is the mesh. The following results were achieved:



N^x	N^y	DIM	$\#^s$	$\#^h$	$\#^{hc}$	$\#^{hc}_S$
2	8	40606	1131	1557	976	976
3	7	36894	551	548	328	328
4	6	35230	305	241	152	152
5	5	34782	224	222	142	142
6	4	35326	307	230	148	148
7	3	37134	531	484	317	317
8	2	41110	1045	1398	912	912

Figure 5: Beam on 16 processors

Although ν varies from 1:1 to 1:64, we have $\#^{hc} = \#^{hc}_S$ in all cases. Probably this is due to the anyhow uniform size and shape of all elements, and an extra benefit of the better coarse grid matrix will occur for nonuniform meshes or nonrectangular elements. Fig. 5 shows, that the additional crosspoint preconditioning decreases the number of iterations in each specified case ($\#^{hc} < \#^h$) and that the **SCCG** as well as (4.3) are more stabel against bad meshes than (3.5).
For the wedge problem we got

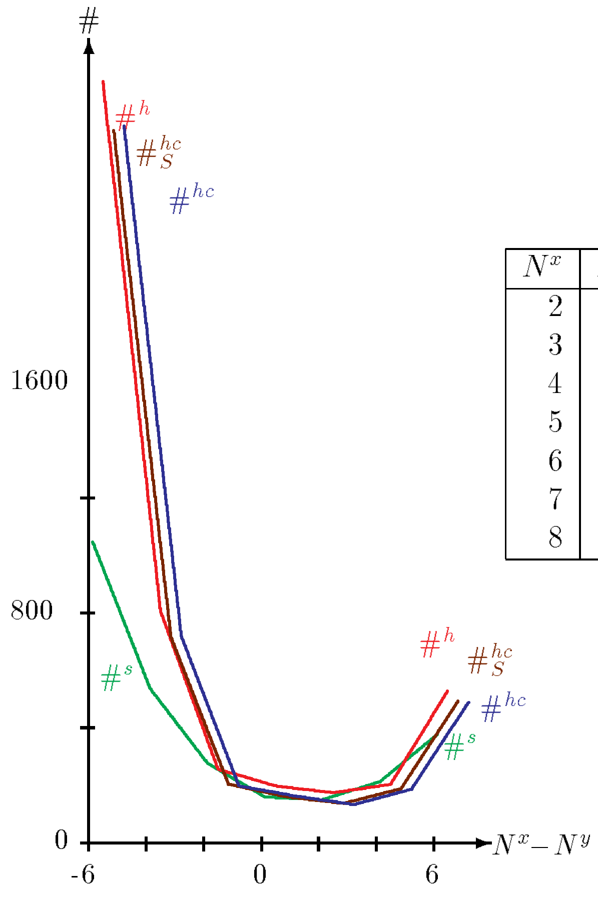


N^x	N^y	DIM	$\#^s$	$\#^h$	$\#^{hc}$	$\#^{hc}_S$
2	8	39064	412	1595	1553	1553
3	7	36120	215	455	428	441
4	6	34840	115	138	132	137
5	5	34584	80	99	88	91
6	4	35224	91	117	96	99
7	3	37080	160	225	199	205
8	2	41080	313	724	665	677

Figure 6: Wedge on 16 processors

Only the **SCCG** proves to be stabel, when the mesh tends to degenerate, whereas the other three solvers show a similar behaviour among each other. Throughout the table (figure 6), we have $\#^{hc} \leq \#^{hc}_{S} < \#^{h}$, emphasizing the usefulness of coarse grid preconditioning and the better features of $\mathbf{L}_Q$ in comparison to $\mathbf{L}_S$.

Finally the membrane problem:
Astonishing we observe, that, in some cases, $\#^{hc}_{S} < \#^{hc}$, but with only small differences. The preconditioner (4.3) was always better than (3.5), and again **SCCG** was well suited for badly shaped meshes.



N^x	N^y	DIM	$\#^s$	$\#^h$	$\#^{hc}$	$\#^{hc}_S$
2	8	39064	1043	2646	2491	2476
3	7	36120	532	801	713	711
4	6	34840	271	251	193	199
5	5	34584	154	193	157	155
6	4	35224	145	169	128	132
7	3	37080	208	198	181	183
8	2	41080	373	526	486	489

Figure 7: Cook's membrane on 16 processors

Summary
In all examples of this section, extra coarse grid preconditioning to the hierarchical preconditioned conjugate gradient method reduces the number of performed iterations, even on geometrical awkward meshes. The difference between the two compared crosspoint matrices is humble, but the real geometry in $\mathbf{L}_Q$ tends to be better than $\mathbf{L}_S$. The number of required iterations grows rapidly, when the meshes degenerate, espacially when hierarchical methods are used. The reason for that behaviour might be, that in hierarchical methods not only the ill conditioned FE–matrix influences the solution process but also the preference of one coordinate direction over the other, especially in such problems, where the unbalance of the mesh doesn't coincide with the simulated physical problem and its boundary conditions (see the unsymetric graphs in figs. 7 and 6).

7.3 Comparison of the preconditioners

After investigating the influence of mesh design, we restrict in the sequal to grids with $N^y = N^x \pm 1$ or $N^y = N^x$. So the number of required iterations is primely depending on the problem and its size[11]. All stated times are in seconds. A ?-mark indicates, that the problem exceeded the available memory.

One single processor

On one single processor no communication occurs and therefore $T_a = T_t$ holds. The following results were measured:

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T^s	T^h	T^{hc}
beam	40	31	59	56	0.04	**0.01**	**0.01**
	144	76	270	247	**0.16**	0.43	0.44
	544	152	857	796	**1.59**	5.00	4.27
	2112	193	1432	1273	**15.63**	29.17	24.54
	8320	208	1610	1432	132.59	125.50	**113.50**
	16512	?	1613	1434	?	252.45	**224.09**
wedge	286	19	43	43	0.15	**0.11**	0.14
	1086	22	57	56	1.23	0.66	**0.64**
	4222	25	70	71	14.82	**3.50**	**3.50**
	8318	25	97	98	30.29	9.57	**9.46**
	16638	?	87	89	?	17.76	**16.90**
membrane	40	22	38	40	0.04	**0.01**	**0.01**
	80	24	44	45	0.07	**0.04**	**0.04**
	144	33	79	77	0.25	0.13	**0.09**
	544	40	115	114	**0.56**	0.64	0.63
	2112	46	142	136	4.50	3.34	**3.10**
	8320	51	156	159	44.89	**14.77**	14.87
	16512	?	162	163	?	**30.33**	30.52

Table 2: Results on 1 processor

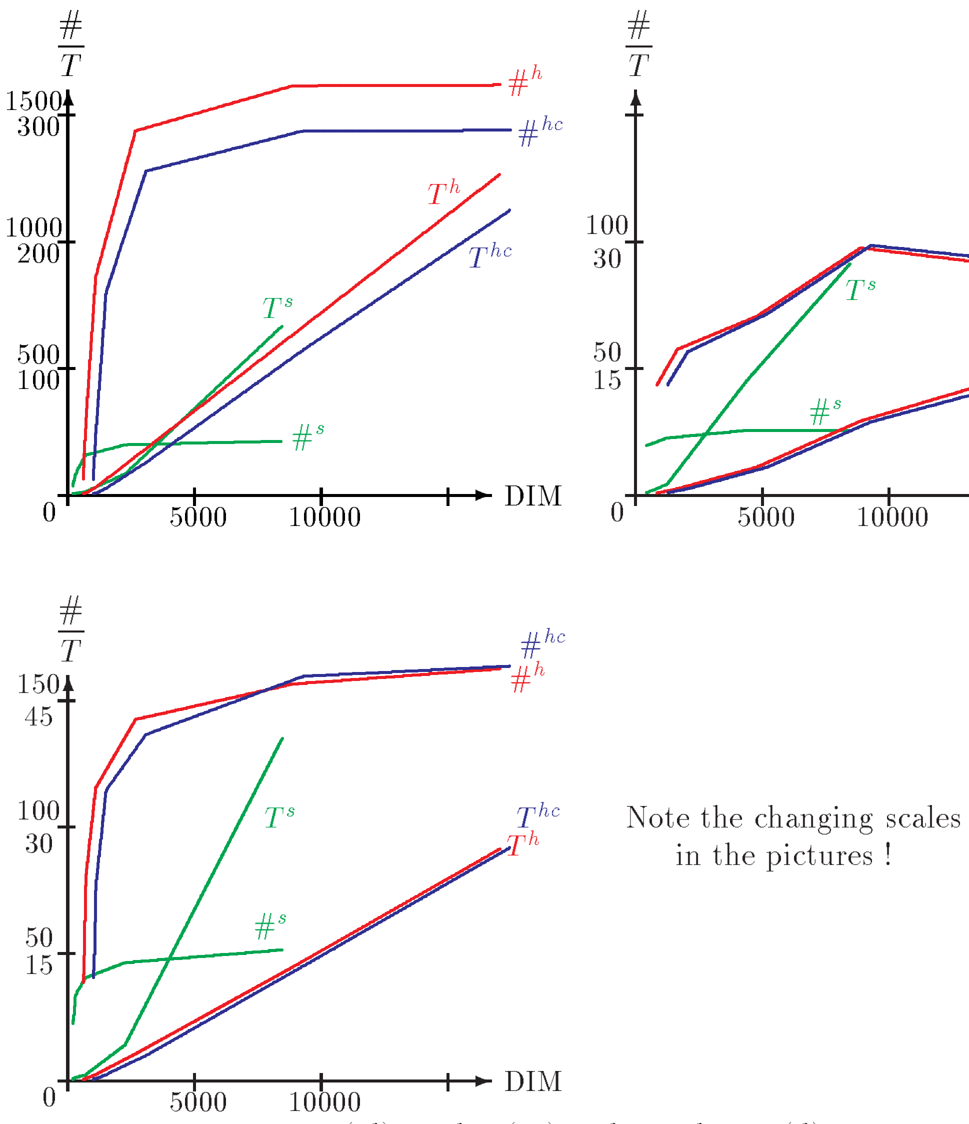
From the data in table 2 and its graphical representation in figure 8, the following predicates can be derived:

1. Due to the low storage requirements of both hierarchical preconditioners, this solvers can handle problems of at least double the size, the **SCCG** is able to solve.

2. The extra coarse grid preconditioning gives a little advantage in iterations and time only in case of the beam problem, whereas for the wedge and the membrane no signifficant differences could be observed.

[11]Remember, that the graphs in figs. 5 to 7 are nearly horizontal for $|N^x - N^y| \leq 1$.

3. Even though both hierarchical preconditioners, at least for a small number of processors, require substantial more steps of iteration than the **SCCG**, the total time for solving the problem with **SCCG** grows much faster with the dimension than the time spend with (3.5) or (4.3). So, for sufficiently large problems, the hierarchical concept proved to be faster.



Note the changing scales
in the pictures !

Figure 8: Beam (ul), wedge (ur) and membrane (d) on 1 processor

Two processors

On two processors, only few communication occurs and therefore $T_a \approx T_t$ holds. Due to the similar behaviour of the solvers for the membrane and the wedge problem (see fig. 8), the results on two processors are presented for the beam and the wedge problem only.

The data in table 3 and the graphics in figure 9 reverify the observations 1 – 3 made in case of one processor.

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T^s	T^h	T^{hc}
beam	90	61	141	132	**0.09**	0.11	0.17
	162	86	168	149	**0.22**	0.26	0.47
	170	98	224	207	**0.21**	0.35	0.54
	306	123	422	381	**0.49**	0.85	1.21
	1122	162	713	583	**2.30**	4.33	3.67
	4290	177	804	659	15.78	18.93	**15.13**
	16770	185	848	693	133.97	78.52	**68.29**
	33282	?	852	695	?	155.27	**128.93**
wedge	304	23	43	44	**0.08**	0.12	0.13
	1120	28	57	58	0.47	**0.41**	0.47
	2176	31	68	70	0.94	**0.92**	1.00
	4288	32	70	75	3.87	**1.88**	2.22
	8448	36	87	93	8.10	**4.79**	5.05
	16786	35	87	94	39.38	**9.58**	10.37
	33408	?	114	124	?	**26.09**	27.53

Table 3: Results on 2 processors

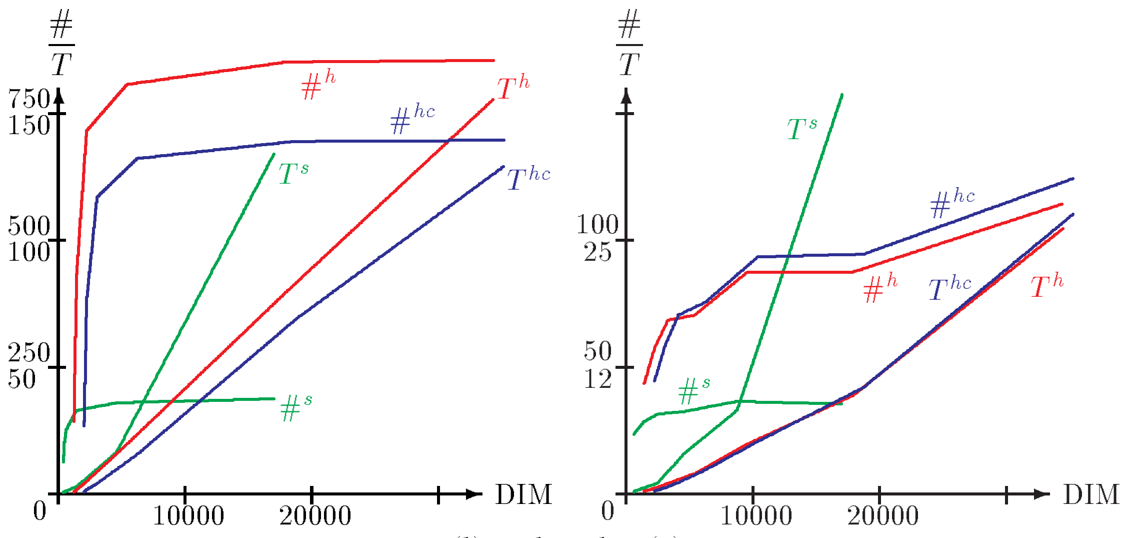


Figure 9: Beam (l) and wedge (r) on 2 processors

Four processors

On four processors, the communication starts to influence the total time consumption of the solver and therefore $T_a \approx T_t$ holds no longer. Again we restrict to the beam and the wedge problem:

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T^s_t	T^s_a	T^h_t	T^h_a	T^{hc}_t	T^{hc}_a
beam	190	101	211	168	**0.41**	0.22	0.55	0.21	0.63	0.15
	630	137	355	256	**0.85**	0.50	1.48	0.85	1.33	0.56
	2278	153	407	291	3.38	2.07	3.24	2.61	**2.75**	1.79
	8646	167	429	309	15.78	15.29	11.11	10.19	**8.08**	7.11
	33670	183	456	323	131.33	130.09	43.87	42.58	**31.57**	30.24
	66822	?	458	326	?	?	86.13	83.86	**60.23**	58.15
wedge	340	28	41	38	**0.12**	0.06	0.15	0.06	0.19	0.08
	1188	33	53	48	0.44	0.36	**0.34**	0.24	**0.34**	0.19
	4420	39	62	57	2.43	2.34	1.01	0.87	**0.96**	0.80
	17028	43	73	68	21.78	21.65	4.17	3.96	**4.00**	3.81
	33540	46	97	91	47.29	46.99	11.05	10.78	**10.57**	10.17
	66820	?	83	78	?	?	19.16	18.84	**17.90**	17.62

Table 4: Results on 4 processors

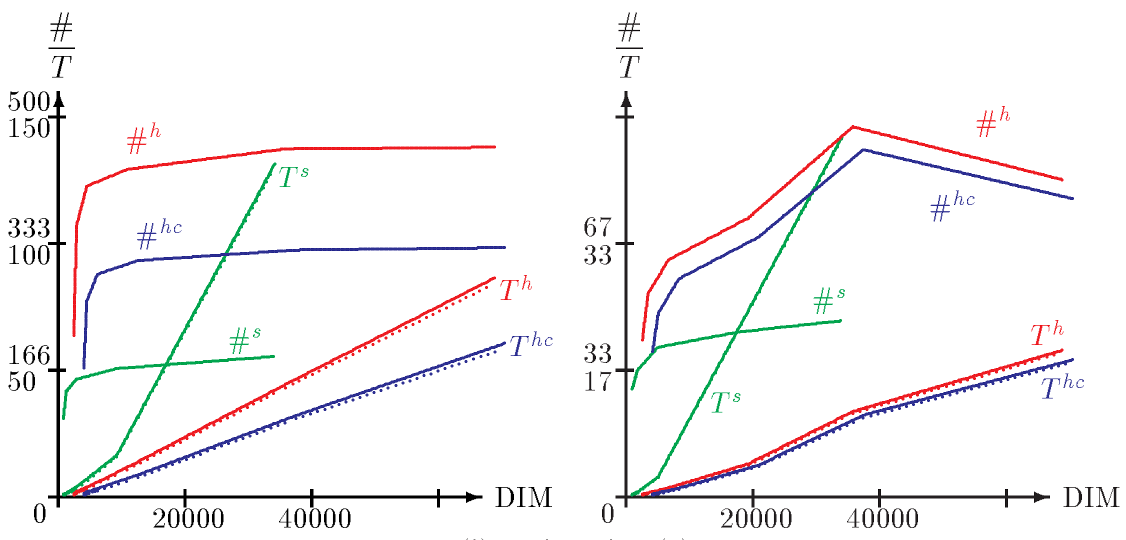


Figure 10: Beam (l) and wedge (r) on 4 processors

Over again, the hierarchical preconditioned solvers can handle problems of double the maximum size, the SCCG-algorithm is able to solve on the same storage basis. Since this remains true for all investigated numbers of processors, it wont't be mentioned anymore in the sequal.
In both problems, the extra coarse grid preconditioner in (4.3) has saved some iterations, compared to (3.5), but it takes nearly double the number of iterations compared to (2.1). Regarding the time, (2.1) ist faster than (3.5) or (4.3) only for small problems, and the more the dimension grows, the larger are the savings gained by using one of the hierarchical preconditioners. In difference to one ore two processors, (4.3) is faster than (3.5) in case of the wedge problem too, provided the problem is large enough.

Since $\#^h - \#^{hc}$ is much smaller for the wedge than for the beam, the time savings $T^h - T^{hc}$ are wee for the wedge but perceptible for the beam.
The tiny differences $T_t - T_a$ show, that the losses in time caused by the different types of global assembly don't play a significant role on 4 processors.
Note that the leftmost segments of the graph related to the number of iterations for the wedge problem in figs. 10 and 8 doesn't indicate a falling tendency. In this cases, the largest resolvable problem was on a mesh with $N^x \neq N^y$ and casual this mesh was anomalous well suited for the solver. The same applies to fig. 12.

Eight processors

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T_t^s	T_a^s	T_t^h	T_a^h	T_t^{hc}	T_a^{hc}
beam	390	121	190	135	**0.39**	0.29	0.54	0.17	0.90	0.28
	1278	143	220	152	**0.70**	0.43	0.90	0.43	1.12	0.37
	4590	167	243	166	2.75	2.36	2.16	1.51	**1.86**	1.00
	17358	186	259	180	19.46	18.86	6.84	6.11	**5.16**	4.34
	67470	206	286	189	148.08	146.94	28.31	27.45	**18.93**	17.87
	133902	?	283	187	?	?	52.95	51.14	**36.30**	34.99
wedge	380	35	40	36	0.19	0.05	**0.18**	0.02	0.22	0.03
	1260	43	52	47	0.55	0.32	**0.35**	0.17	0.36	0.12
	4556	51	62	56	1.24	1.05	**0.74**	0.46	0.76	0.46
	17292	57	74	67	6.16	5.89	2.53	2.13	**2.34**	1.88
	67340	65	85	77	59.26	58.73	10.35	9.89	**9.06**	8.42
	133644	?	102	93	?	?	23.07	21.93	**20.95**	19.87

Table 5: Results on 8 processors

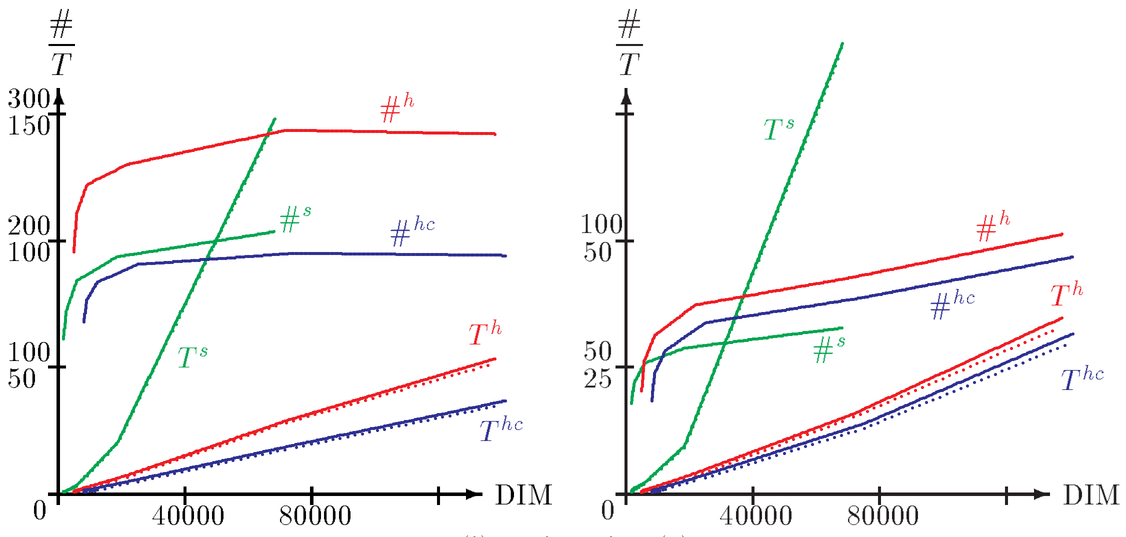


Figure 11: Beam (l) and wedge (r) on 8 processors

On eight processors, for the first time we observed $\#^{hc} < \#^{s}$ yet, at first for the beam problem only. For the wedge problem, $\#^{hc} - \#^{s}$ is at least 12.
The difference in time consumption of the hierarchical preconditioned solvers and the **SCCG** is growing rapidly with the dimension of the problems. For both tasks we have $T^{hc} < T^{h} < T^{s}$ whenever the dimension is large enough.

16 processors

On 16 processors we also present results for the membrane problem from fig. 4:

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T_t^s	T_a^s	T_t^h	T_a^h	T_t^{hc}	T_a^{hc}
beam	790	145	156	107	**0.81**	0.28	0.85	0.22	1.16	0.33
	2574	173	180	120	1.37	0.63	**1.12**	0.49	1.43	0.51
	9214	199	203	132	4.42	3.43	2.39	1.32	**2.02**	0.88
	34782	224	222	142	23.54	22.76	6.41	5.19	**4.79**	3.53
	135070	248	242	155	175.47	174.24	24.65	**23.34**	17.06	15.55
	268062	?	243	149	?	?	47.35	44.57	**30.13**	28.23
wedge	760	49	67	60	**0.39**	0.07	0.48	0.10	0.67	0.19
	2520	59	78	67	**0.63**	0.24	0.68	0.21	0.77	0.19
	9112	70	88	78	1.57	1.09	1.30	0.66	**1.29**	0.62
	34584	80	99	88	9.11	8.71	3.68	2.82	**3.32**	2.55
	134680	84	108	99	71.61	70.84	13.14	12.00	**12.28**	11.13
	267800	?	101	85	?	?	23.35	21.77	**20.21**	19.07
mem–brane	760	99	130	102	**0.77**	0.27	0.98	0.19	1.06	0.30
	2520	120	146	126	1.38	0.55	**1.29**	0.39	1.45	0.35
	9112	138	179	138	3.10	2.13	2.67	1.33	**2.31**	1.10
	34584	154	193	157	16.88	15.59	7.05	5.26	**6.05**	4.57
	134680	172	214	177	127.50	125.71	26.66	24.35	**22.16**	19.86
	267288	?	216	182	?	?	49.94	46.70	**42.81**	40.39

Table 6: Results on 16 processors

Regarding the beam problem, the lowest number of needed iterations was obtained with the hierarchical solver and additional coarse grid preconditioning, followed with a large desistance by the **SCCG** and the pure hierarchical solver. If the dimension of the linear system is large enough, yet the pure hierarchical solver needs slightly few iterations than the **SCCG**.
For the two other tasks we have $\#^{s} < \#^{hc} < \#^{h}$.
Viewing at the time, both variants of the hierarchical preconditioners proofed their supremacy about the **SCCG** in all situations.

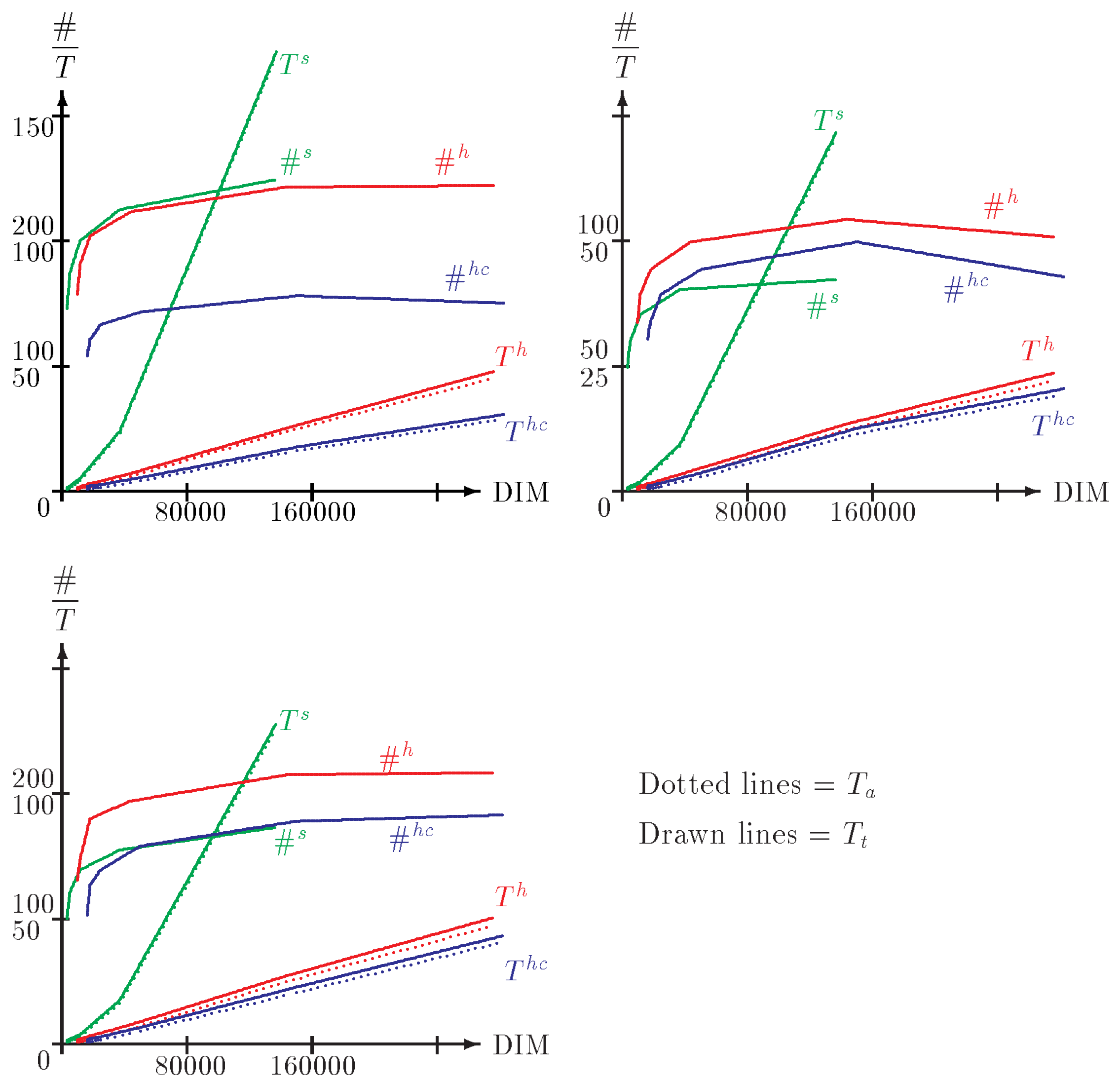


Figure 12: Beam (ul), wedge (ur) and membrane (d) on 16 processors

32 processors

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T_t^s	T_a^s	T_t^h	T_a^h	T_t^{hc}	T_a^{hc}
beam	1590	234	200	103	1.49	0.37	**1.05**	0.23	1.47	0.30
	5166	273	224	113	3.14	2.00	**1.44**	0.48	1.82	0.50
	18462	308	245	122	8.32	7.00	2.91	1.54	**2.43**	1.01
	69630	344	264	134	35.45	33.66	7.67	6.49	**5.12**	3.43
	270270	377	283	144	259.63	253.27	29.89	27.21	**16.60**	13.98
	536382	?	328	161	?	?	65.01	61.72	**34.42**	30.25
wedge	1560	69	58	43	0.75	0.23	**0.55**	0.11	0.68	0.14
	5112	81	68	53	1.19	0.59	**0.73**	0.18	0.98	0.34
	18360	91	78	62	2.44	1.86	**1.39**	0.59	1.49	0.67
	69432	101	90	72	11.81	10.87	3.63	2.56	**3.41**	2.08
	269880	109	102	82	87.47	86.06	13.44	11.43	**11.47**	9.48
	536120	?	127	111	?	?	31.77	29.07	**28.49**	25.15

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T_t^s	T_a^s	T_t^h	T_a^h	T_t^{hc}	T_a^{hc}
	1560	118	104	66	1.15	0.25	**0.97**	0.20	0.99	0.12
	5112	139	121	77	1.94	0.89	1.32	0.31	**1.30**	0.25
mem–	18360	156	140	93	4.08	2.56	2.46	1.09	**2.19**	0.94
brane	69432	173	157	106	18.01	16.70	6.20	4.43	**4.80**	3.27
	269880	188	173	120	143.03	138.68	21.79	19.29	**15.94**	13.72
	536120	?	172	131	?	?	42.55	39.08	**33.41**	29.85

Table 7: Results on 32 processors

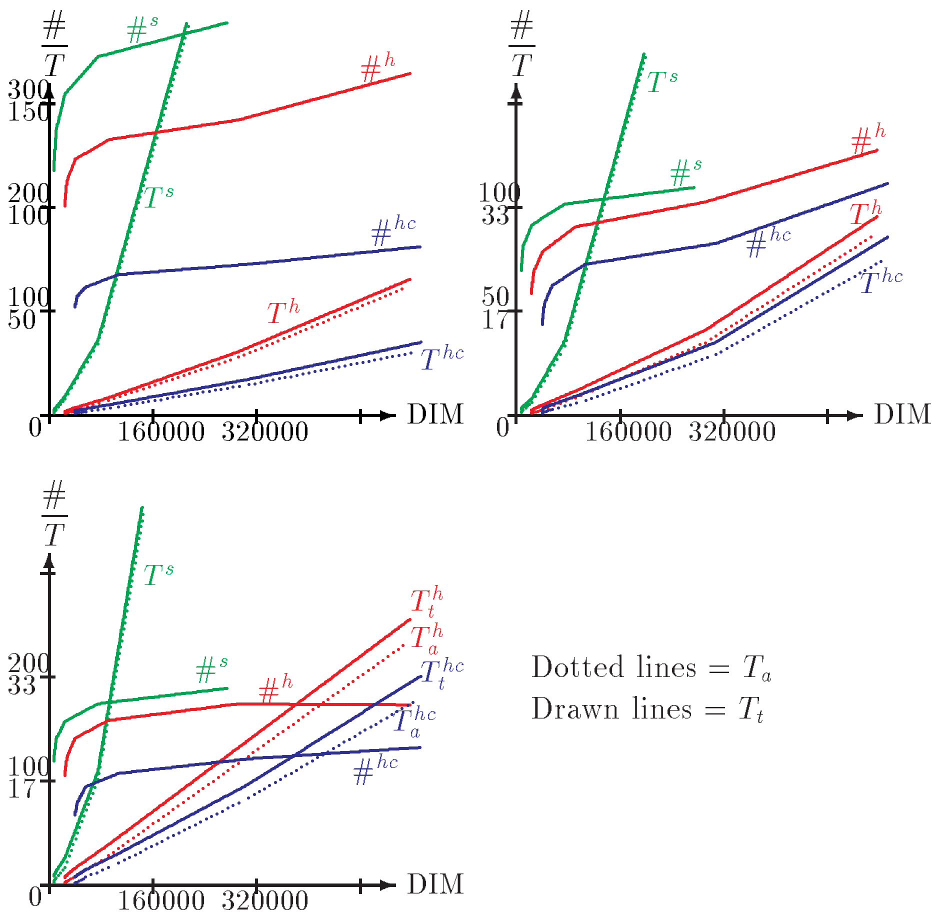


Figure 13: Beam (ul), wedge (ur) and membrane (d) on 32 processors

On 32 processors we observe $\#^s > \#^h > \#^{hc}$ for all three investigatd model problems. Therefore $T^s > T^h$ also holds for all mesh sizes, and except for large mesh sizes and therefore small numbers of unknowns in the linear systems, $T^h > T^{hc}$ holds too.

64 processors

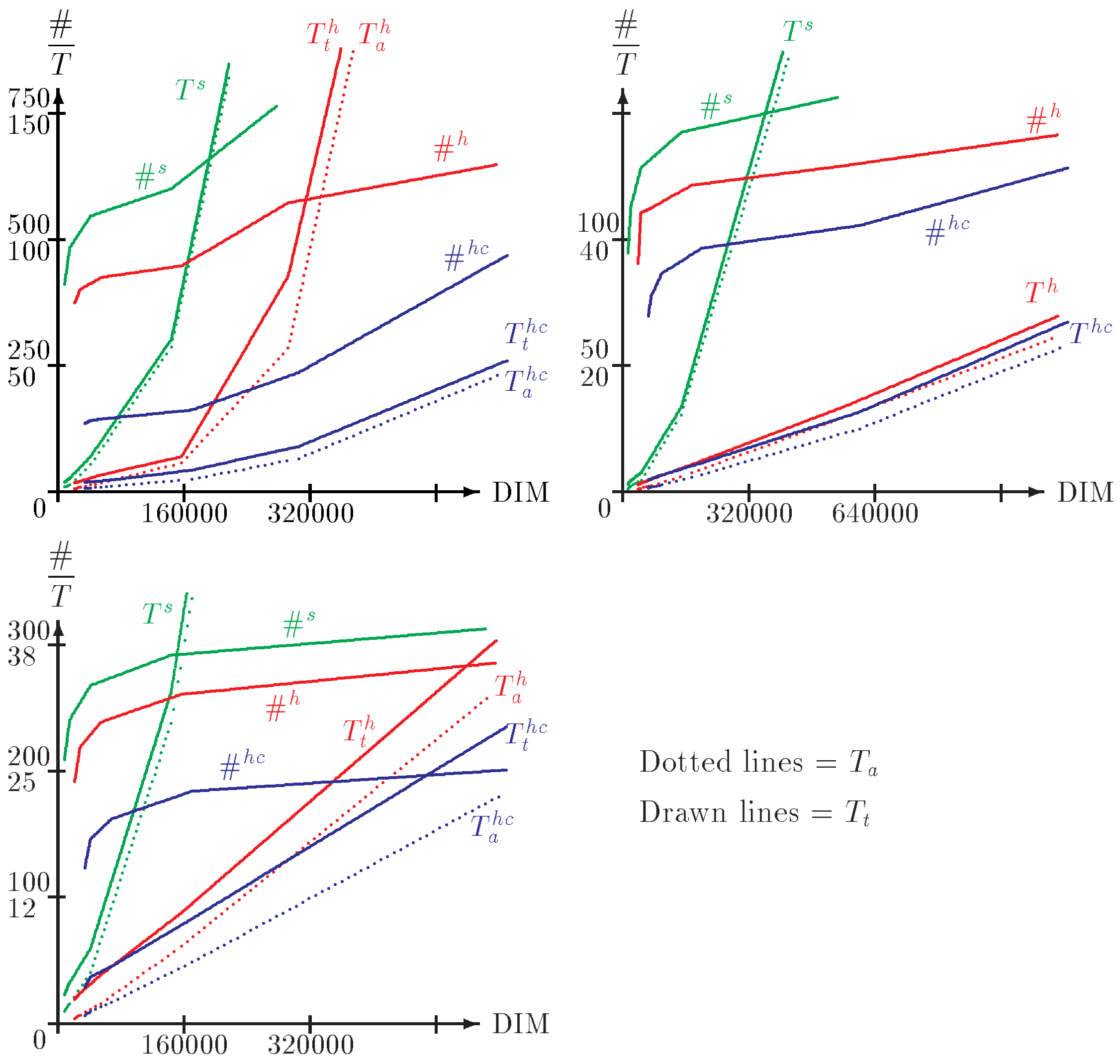


Figure 14: Beam (ul), wedge (ur) and membrane (d) on 64 processors

Again we observe $\#^s > \#^h > \#^{hc}$ and $T^s > T^h$ in all cases, also $T^h > T^{hc}$ for sufficiently large problems.

Task	DIM	$\#^s$	$\#^h$	$\#^{hc}$	T^s_t	T^s_a	T^h_t	T^h_a	T^{hc}_t	T^{hc}_a
	3190	408	371	132	3.16	1.17	**2.71**	0.47	3.10	0.70
	10350	481	398	138	4.75	1.94	3.53	1.10	**3.33**	0.76
beam	36958	544	422	143	13.41	10.03	6.02	2.82	**3.99**	1.39
	139326	598	445	159	59.94	56.98	13.30	10.82	**7.92**	4.31
	274494	764	570	233	261.75	255.07	84.92	56.29	17.32	12.50
	540670	?	646	467	?	?	442.86	438.19	**51.60**	47.28

Task	DIM	$\#^{s}$	$\#^{h}$	$\#^{hc}$	T_t^s	T_a^s	T_t^h	T_a^h	T_t^{hc}	T_a^{hc}
wedge	3120	94	90	69	1.20	0.31	**1.06**	0.16	1.97	0.43
	10224	112	110	77	1.86	0.80	**1.36**	0.31	2.27	0.49
	36720	128	112	86	3.45	1.90	**2.16**	0.85	3.00	1.04
	138864	142	121	96	16.44	14.98	**5.06**	3.41	5.41	3.06
	539760	156	129	105	126.06	119.12	16.98	14.82	**15.52**	12.19
	1071216	?	141	128	?	?	34.58	30.54	**33.50**	28.90
mem-brane	3120	208	191	122	2.69	1.02	**2.24**	0.33	3.44	0.62
	10224	240	218	145	3.92	1.80	**2.85**	0.69	4.45	1.04
	36720	267	238	161	7.32	4.91	**4.67**	1.82	5.50	1.93
	138864	291	260	183	32.73	29.61	10.82	7.48	**10.22**	5.89
	539760	312	285	200	231.42	219.85	37.93	32.93	**29.34**	22.85

Table 8: Results on 64 processors

Summary

The hierarchical preconditioners are more efficient in space and time than the **SCCG**, but require special meshes. The larger the number of processors and the larger the number of unknowns on each processor, the faster in comparison to the **SCCG** is even the poor hierarchical proconditioned CG–algorithm and the larger is the extra gain in time, achieved with the additional coars grid preconditioner.

7.4 Different crosspoint matrices

Some first comparings of the two coarse grid matrices $\mathbf{L}_Q$ and $\mathbf{L}_S$ from section 4.1 on badly shaped meshes were done in section 7.2. In this section, both versions of coarse grid preconditioning shall be compared to each other in more typical situations on 16 and on 32 processors:
If the beam–problem is simulated on 16 processors, all subdomains are squares and hence we have $\mathbf{L}_Q = \mathbf{L}_S$, resulting in an identical behaviour of both versions of the solver. For the other two problems we got the following:

	wedge				membrane			
DIM	$\#^{hc}$	$\#_S^{hc}$	T^{hc}	T_S^{hc}	$\#^{hc}$	$\#_S^{hc}$	T^{hc}	T_S^{hc}
760	60	64	0.67	0.64	102	104	1.06	1.06
2520	67	71	0.77	0.83	126	121	1.45	1.42
9112	78	81	1.29	1.36	138	140	2.31	2.32
34584	88	91	3.32	3.59	157	155	6.05	5.38
134680	99	100	12.28	12.61	177	173	22.10	21.75

Table 9: $\mathbf{L}_Q$ and $\mathbf{L}_S$ on 16 processors

We observe $\#^{hc} < \#_S^{hc}$ for the wedge problem and, with one single exception which

is probably caused by any disturbance of the time measuring routines, therefore also $T^{hc} < T_S^{hc}$.
On 32 processors, the beam–promlem's subdomains are still rectangular but nevermore squares.

	wedge				membrane			
DIM	$\#^{hc}$	$\#_S^{hc}$	T^{hc}	T_S^{hc}	$\#^{hc}$	$\#_S^{hc}$	T^{hc}	T_S^{hc}
1560	43	46	0.68	0.72	66	67	0.99	1.01
5112	53	57	0.98	1.06	77	79	1.30	1.38
18360	62	65	1.49	1.55	93	93	2.19	2.12
69432	72	75	3.41	3.35	106	106	4.80	4.81
269880	82	86	11.49	11.45	120	119	15.94	15.98

	beam			
DIM	$\#^{hc}$	$\#_S^{hc}$	T^{hc}	T_S^{hc}
1590	103	131	1.49	1.87
5166	113	147	1.82	2.35
18462	122	161	2.43	3.33
69630	134	172	5.12	6.55
270270	144	188	16.60	21.04

Table 10: $\mathbf{L}_Q$ and $\mathbf{L}_S$ on 32 processors

With one single exception (membrane problem with 269880 degrees of freedom) $\#^{hc} < \#_S^{hc}$ holds, and $T^{hc} < T_S^{hc}$ is valid in all cases. Especially for the beam problem we observed the largest differences $T_S^{hc} - T^{hc}$.
Summarizing, the crosspoint matrices $\mathbf{L}_Q$ basing on the real geometry have prooven to be the better choice.

References

[1] G.Haase,T.Hommel,A.Meyer und M.Pester, Bibliotheken zur Entwicklung paralleler Algorithmen, Preprint Nr. SPC 94_4, Fakultät f. Mathematik, TU Chemnitz-Zwickau

[2] M.Meisel, A.Meyer, Implementierung eines parallelen vorkonditionierten Schur-Komplement CG-Verfahrens in das Programmpaket FEAP, Preprint Nr. SPC 95_2, Fakultät f. Math., TU Chemnitz–Zwickau, Januar 1995

[3] M.Meisel, A.Meyer, Kommunikationstechnologien beim parallelen vorkonditionierten Schur–Komplement CG–Verfahren, Preprint Nr. SPC 95_19, Fakultät f. Math., TU Chemnitz–Zwickau, Juni 1995

[4] A.Meyer, A Parallel Preconditioned Conjugate Gradient Method Using Domain Decomposition and Inexact Solvers on Each Subdomain, Computing 45, 1990.

[5] S.Meynen und P.Wriggers, Tätigkeitsbericht Darmstadt zum Forschungsvorhaben Wr19/5-1 Parallele Iterationsverfahren, Technische Hochschule Darmstadt, Institut für Mechanik, Juli 1994

[6] R.L.Taylor, FEAP - A finite element analysis program, Description and Users-Manual, University of California, Berkeley,1990

[7] H.Yserentant, On the multilevel–splitting of finite element spaces, Numer. Math. 49, 379–412 (1986)