Proceedings of ALGORITMY 2000
Conference on Scientific Computing, pp. 388–398

# LIBRARY OF PARALLEL PCG SOLVERS FOR PROBLEMS OF GEOMECHANICS

RADIM BLAHETA, ONDŘEJ JAKL AND JIŘÍ STARÝ*

**Abstract.** The paper describes two ways of parallelization, based on the displacement decomposition and domain decomposition, of the preconditioned conjugate gradient method, which is used for the construction of solvers of the GEM32 finite element package. The crucial points of the algorithms, especially the preconditioning operation, are exposed. The second part of the paper reports the benefits of the parallelization for the solution of some real problems, including an example of large scale mathematical modelling in geomechanics.

**Key words.** preconditioned conjugate gradient method, parallelization, displacement decomposition, domain decomposition, incomplete factorization, Schwarz methods, large scale computations

**AMS subject classifications.** 65F10, 65N22, 65Y05

**1. Introduction.** Mining geomechanics is not an exception in extended application of mathematical modelling. On the contrary, it is a rich source of complex phenomena to be modeled, such as development of mine openings, effects caused by inelastic and damage behaviour of geomaterials or actions of joints and supports, to name some objects of practical interest. Deformation and stress fields are usually expected as the result of the modelling, often in multiple variants for determining the influence of uncertain input parameters. From the mathematical point of view, the problems are formulated as two- and three-dimensional partial differential equations as used in elasticity and plasticity. For their numerical solution, *finite element (FE)* analysis is mostly applied, resulting in large linear or nonlinear algebraic systems. Their handling is the crucial and most time-consuming part of the solution process, for which efficient techniques, including parallel approach, are intensively studied.

The *GEM32* software package, continuously developed at the Institute of Geonics of the Czech Academy of Sciences both for research purposes and practical modelling, is an example of a computer code addressing 3-D FE analysis of elasticity and plasticity problems arising in geomechanics. Its characteristics include discretization of the modeled domain using *regular structured grids*, which result from deformation of a regular rectangular 3-D grid of nodes, and iterative solvers based on the *preconditioned conjugate gradient (PCG)* method.

The growing demand for the solution of complex problems arising in the geomechanical practice accelerated the development of GEM32 towards large scale modelling. Here especially, the critical issue is the solution of large systems of linear systems, often of millions of unknowns. With increasing availability of the multiprocessor hardware, a promising direction towards the required performance of the PCG solver could be seen, among other things, in its parallelization.

This article describes the work which the authors have done in this respect. It presents the basic methods applied in that transition and resulting codes which in fact form a collection or *library* of solvers based on the PCG method, displacement

*Academy of Science of the Czech Republic, Institute of Geonics, Studentská 1768, 708 00 Ostrava – Poruba and VŠB – Technical University, 17. listopadu 15, 708 33 Ostrava – Poruba ({blaheta|jakl|stary}@ugn.cas.cz)

decomposition and overlapping domain decomposition.

**2. Sequential PCG solver.**
We shall restrict our attention to the solution of problems of 3-D linear elasticity by the FE method with linear tetrahedral finite elements, as it is practised in GEM32. In the preprocessing stage of modelling, GEM32 must be fed in with the (1) geometry of the 3-D domain of interest and its discretization by a regular structured grid, (2) loading given by the weight of the material and action of the surrounding rock massif and (3) specification of the material properties and material distribution. This data is used for assembling the FE system of linear equations

$$Au = b \tag{2.1}$$

where $A$ is symmetric, positive definite $n \times n$ *stiffness matrix*, $b \in R^n$ is the right-hand side given by the loading and $u \in R^n$ is the $n$-dimensional vector of unknown nodal displacements. The linear system is then forwarded to the GEM32 solver.

As mentioned above, GEM32 solvers make use of the well-known PCG method. Its algorithm is outlined in Fig. 2.1. In this scheme (where S and E respectively denote the start and end of the procedure), $\hat{u}$ is an *initial approximation*, $r \in R^n$ the *residual*, $w, v \in R^n$ auxiliary vectors, $\alpha, \beta, s_0, s_1 \in R$ scalars and $P$ the *preconditioning operator* (discussed below). The *termination criterion* TC has the form $||r|| = \varepsilon\, ||b||$, where $\varepsilon$ is the required *relative accuracy* (usually $10^{-3}$ – $10^{-5}$).

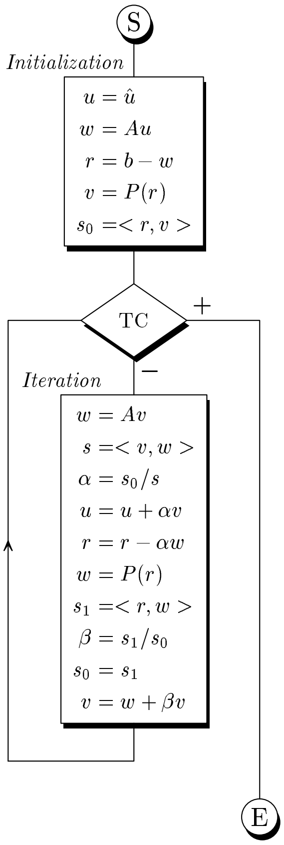


FIG. 2.1. *The PCG method*

**2.1. Preconditioning.** In the PCG algorithm, preconditioning is a very important step which, roughly speaking, improves the search directions for the solution. GEM32 provides preconditioning based on the so-called *displacement decomposition (DiD)*, studied in [1] and [2]. DiD separates the three components of the nodal displacement vector (in the $x, y, z$ directions) by rearranging and decomposing the vector $u$ (and correspondingly the matrix $A$ of the system (2.1)) as follows:

$$u = (u_1, u_2, u_3)$$
$$A = (A_{kl}) \qquad k, l = 1, 2, 3$$

Here, blocks $u_1, u_2, u_3$ respectively represent nodal displacements in $x, y, z$ directions.

Having this, GEM32 can offer two preconditioning techniques. The first one, called *incomplete factorization (DiD-IF)*, results from the incomplete factorization of the matrix $\tilde{A}$ defined as follows:

$$\tilde{A}_{kk} = A_{kk} \qquad k = 1, 2, 3$$
$$\tilde{A}_{kl} = 0 \qquad k, l = 1, 2, 3,\ k \neq l$$

In [1], [2] it is shown that:

1. Matrix $\tilde{A}$ is spectrally equivalent to $A$.
2. Incomplete factorization of blocks $A_{kk}$ is stable in the case of linear tetrahedral finite elements and reasonable requirements on the division of the domain into tetrahedrons. (These requirements are fulfilled by most of the applications).

The preconditioning operation $w = P(r)$ is then given by the (inexpensive) solution of the system $Cw = r$, where the (block diagonal) preconditioning matrix $C$ has diagonal blocks of the form $C_{kk} = (X_k + L_k){X_k}^{-1}(X_k + L_k)^T$. Here, $L_k$ is the strictly lower triangular part of the matrix $A_{k,k}$, $X_k$ is a diagonal matrix determined by the condition of equal rowsums of the matrices $C_{kk}$ and $A_{kk} + \tilde{D}_k$ for an appropriate diagonal perturbation matrix $\tilde{D}_k$. For details see [2].

DiD-IF is the traditional preconditioner for the GEM32 solvers. As we shall see in the next section, it has favorable properties for parallelization.

Motivated also by the parallelization, an alternative to the incomplete factorization preconditioning has been introduced recently. This *variable preconditioning* (denoted *DiD-VP)* defines the $w = P(r)$ operation as a low accuracy solution of the system $\tilde{A}w = r$ by another, *inner* PCG iteration process, which can again use the DiD-IF preconditioning. While being of limited benefit for the sequential solution, it turned out to be very useful in the parallel case, because it migrates the computational work to communication-free inner iterations (cf. Section 6.3). For more mathematical issues see [5], [4].

**3. Parallelization based on displacement decomposition.** The decomposition of the vector $u$ into (in general $m$) equally sized blocks is also a natural step towards the parallelization of the PCG algorithm. Following the common data decomposition paradigm and "single program – multiple data" (SPMD) parallelization model, the solution of the whole system (2.1) can be then assigned blockwise to $m$ tasks which then perform the PCG algorithm concurrently (with some data exchange) on their portion of data: The $i$-th task processes the block $u_i$ of the displacement vector and the corresponding column block $A_i = \{A_{ji}; j = 1, \ldots, m\}$ of the stiffness matrix. For synchronization and supervision purposes, it is practical to accompany these *worker* tasks by a *master* task, which combines partial results to global values, evaluates the termination criterion, etc. The scheme of such a parallel algorithm is presented in Fig. 3.1.

There are two particular points, where this straightforward parallelization procedure may bring difficulties. The first one (denoted $\mathcal{M}$ in Fig. 3.1), is the matrix-vector product, which requires extensive intertask communication $\mathcal{C}_{\mathcal{M}}$. Namely: The $i$-th worker computes the products $w_{ij} = A_{ji}v_i$, $j = 1, \ldots, m$, and transfers all but the $w_{ii}$ value to other workers ($w_{ij}$ to the $j$-th worker). On the contrary, it must obtain their $w_{ji}$, $j \neq i$, to be able to calculate the new value of $w_i = \sum_j w_{ji}$.

The second point (denoted $\mathcal{P}$), even more subtle, is the preconditioning operation. Here we can in general observe only that if the operation has the form of $Cw = r$, an efficient parallelization seems to be possible provided that $C$ is block diagonal. In this situation, no task interaction $\mathcal{C}_{\mathcal{P}}$ is necessary, but this is not a general case.

Note that the other master – worker communications/synchronizations in the iterative part of the parallel PCG algorithm can be reduced by one when applying the *Chronopoulos-Gear algorithm* with a modified expression for the search parameter $\alpha$ (see [7]). The same effect can be attained by using the standard formula for $\alpha$, but a block row-wise partitioning of the stiffness matrix, allowing some rearrangements of the operations; see [6].

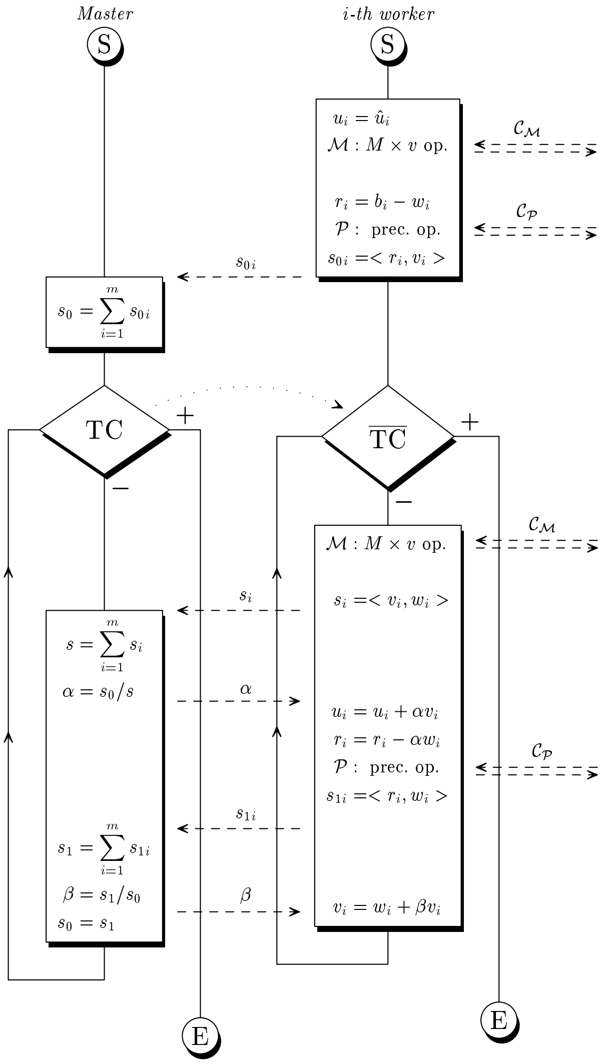


FIG. 3.1. *Parallelized PCG algorithm – a general scheme*

Let us now apply this general framework to the case of displacement decomposition. From the nature of DiD it follows that $m = 3$ is fixed, i.e. there are exactly three worker tasks cooperating on the solution. At the point $\mathcal{M}$, these tasks have to exchange six vectors, having $n/3$ real components each. For systems with e.g. one million unknowns this means transfer of eight megabytes of data in every iteration. That is why this operation requires a careful technical realization, to minimize the overhead.

Preconditioning $\mathcal{P}$ is the point where DiD shows its advantage. As we could see in the previous section, both DiD-IF and DiD-VP preconditioners construct the preconditioning matrix $C$ as a block diagonal matrix, which can be fully decomposed. Thus, the workers can complete this operation independently, without need of any communication, just as in the sequential case.

Although the DiD-based parallelization can considerably shorten processing time of some mathematical models (cf. Section 6.3), its general applicability is strongly limited by the fixed (and low) number of parallel tasks inherent in its fundamentals, which rules out scalability (i.e. the ability to utilize a greater number of processors). For this reason, in the next phase we focused our attention on the *domain decomposition (DD)* methods.

**4. Parallelization based on domain decomposition.** DD methods are based on the geometrical partitioning of the domain of interest into subdomains. In principle, they do not limit the number, shape and size of subdomains, being thus highly scalable. For a comprehensive treatment of these methods in the context of parallel computing see for example [9].

The initial DD implementation in GEM32 considers a simple one-dimensional partitioning of the domain $\Omega$, already discretized with a regular structured grid, in the $z$ direction to $m$ disjunct subdomains $\bar{\Omega}_1, \ldots, \bar{\Omega}_m$, so that each grid node belongs to exactly one subdomain. To improve the convergence however, we can make subdomains partially overlap: Every $\bar{\Omega}_i$ is extended to $\Omega_i$ to share some "layers" of nodes with the adjacent subdomain $\bar{\Omega}_{i+1}$ (let us call the number of shared layers *overlapping factor*). Fig. 4.1 illustrates this setup for $m = 3$ and overlapping factor 1.

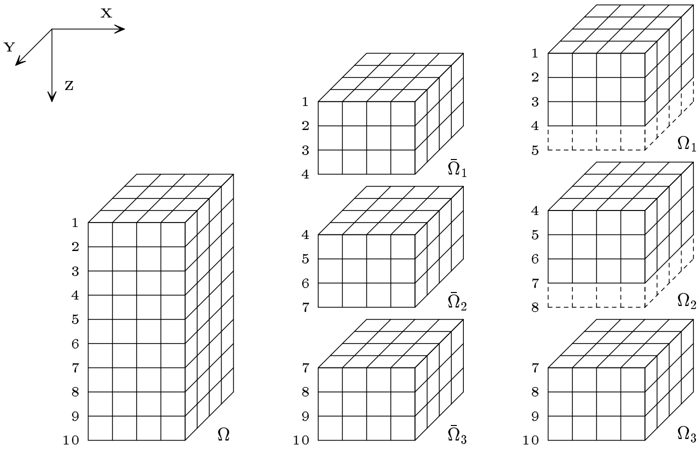


FIG. 4.1. *Domain decomposition of a regular structured grid*

Having this partition, it is possible to use iterative methods and preconditioners referred to as *Schwarz methods*. The computation is decomposed by associating parallel tasks with subdomains. Each subdomain is processed by one of the $m$ concurrent tasks. Interactions are necessary only between "neighbours" in the 1-D decomposition.

From the point of view of the system (2.1), the $i$-th task processes the stiffness matrix $A_i$, loading vector $b_i$ and displacement vector $u_i$ (and related auxiliary structures), which are associated with the subdomain $\Omega_i$. These blocks are extracted from the original system by copying those rows that correspond to the nodes of the subdomain $\Omega_i$.

The formal scheme of the DD algorithm is the same as in the DiD case, see Fig. 3.1. Let us focus on the points $\mathcal{M}$ and $\mathcal{P}$, where the differences are hidden.

In $\mathcal{M}$, a standard matrix-by-vector multiplication $\bar{w}_i = \bar{A}_i \bar{v}_i$ is performed (on the non-overlapping part of $A_i$). To keep the vectors $w_i$ updated for all rows of $A_i$ and consistent with the global operation $w = Av$, pairs of neighbouring tasks $(i, i-1)$ have to exchange calculated components related to the shared rows ($\mathcal{C}_\mathcal{M}$ communication). The amount of this data transfers depends on the bandwidth of $A$ and the density of the discretization in the $x$ and $z$ directions.

To explain the preconditioning point $\mathcal{P}$, we introduce the matrix $R_i$ representing the restriction operation $R_i : u \to u_i$, where $u_i$ is the vector of those components of $u$ that correspond to the nodes from the closure of the subdomain $\Omega_i$. The preconditioning operation employs the so called *additive Schwarz preconditioner* and has the following form:

$$w = \sum_{i=1}^{m} R_i^T A_i^{-1} R_i r \tag{4.1}$$

The workers solve their systems in analogy with Section 2.1 by replacing $A_i$ by its incomplete factorization (in practice preferred) or by inner PCG iterations of the variable preconditioning. Thereafter, the neighbours are required to exchange (and sum) results for the shared rows (within the $\mathcal{C}_\mathcal{P}$ communication).

**4.1. Application of the coarse grid.** The Schwarz method can be made more efficient if it can take advantage of a "global" information, represented by a solution of the same problem, but discretized by a coarser grid. The idea is that for the coarse grids we assemble the stiffness matrix $A_c$ with regard to the principal boundary conditions. Let $R_c$ is the matrix of linear interpolation transforming the values from the fine to the coarse grid. Then we can define the *two level additive Schwarz preconditioner* by the expression

$$w = (P_c + \sum_i R_i^T A_i^{-1} R_i) r \tag{4.2}$$

where $P_c = R_c^T A_c^{-1} R_c$ is the *coarse grid part* of the preconditioner.

An alternative to the additive approach is to include the coarse grid part of the preconditioner in a multiplicative way. This can be achieved by the formula

$$w = P_c r + \sum_i R_i^T A_i^{-1} R_i (r - AP_c r) \tag{4.3}$$

called *two level hybrid non-symmetric Schwarz preconditioner*. Note that this preconditioner can be symmetrized, but good results have been obtained even with this non-symmetric version.

It is natural to realize the coarse grid computation as a stand-alone worker task which runs in parallel with other workers processing the subdomains. Whereas in the additive case this task can run completely in parallel with the subdomain workers, in the latter case the subdomain workers have to wait for the coarse grid worker to send $P_c r$, to be able to complete the preconditioning operation. Nevertheless with sufficiently large grid, the benefits of the multiplicative approach to the convergence may outweigh this drawback.

**5. Computer realization.** The parallel algorithms described in previous sections are implemented as experimental solver modules of the GEM32 package. Their Fortran code has its origin in the sequential solver *PCG-S* [3] and wherever possible, they make use of its data files, data structures and supporting utilities.

The parallel algorithms are designed to match the universal *message passing model*, where task interaction is accomplished by exchanging messages, with no requirements on shared memory. For the realization of message passing, we chose the

*Parallel Virtual Machine (PVM)* [8] API, but some experiments with the *Message Passing Interface (MPI)* implementations have been performed as well. Since these message passing systems seem to be available on all parallel architectures, the solvers should be portable to most platforms, from supercomputers to clusters of workstations.

Both the methods and the codes are continuously developed and experiments are in progress. In the next section, we selected some examples to give an idea about the achieved performance. The presented codes are characterized in Tab. 5.1.

| *Solver name* | *Type* | *Decomposition* | *Preconditioning* |
|---|---|---|---|
| PCG-S | seq. | – | incomplete factorization |
| PCG-P(DiD-IF) | PVM | displacement | incomplete factorization |
| PCG-P(DiD-VP) | PVM | displacement | variable |
| PCG-P(DD) | PVM | domain | 2-lev. hybrid non-symm. Schwarz |

TABLE 5.1
*Main characteristics of the solvers*

**6. Experiments and results.** This section informs about numerical experiments performed with the parallel solvers. Particularly in the DD case the testing is still in progress.

**6.1. Benchmarks.** One of the standard benchmarks that we use for evaluating the performance of solvers is the so called *square footing problem (FOOT)*, a 3D elasticity problem of soil mechanics (Fig. 6.1). It deals with the influence of flexible footing represented by localized pressure (on the top side of the domain) to the stress development in a soil medium. Due to symmetry, the computational domain $\Omega$ is restricted to a quarter of the whole situation. The standard discretization is accomplished by a rectangular grid of $40 \times 40 \times 40$ nodes, with grid refinement under the footing, providing a linear system of 192 000 equations. Using a denser grid, this relatively small benchmark (FOOT40) can be made larger (FOOT$n$).

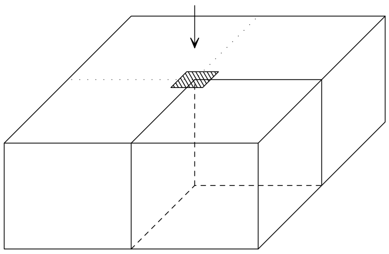

FIG. 6.1. *The FOOT benchmark*

As an example of a large scale computation, let us introduce a practical problem related to mining in the uranium ore deposit at *Dolní Rožínka (DR)* in the Bohemian-Moravian Highlands. This model considers a domain of $1430 \times 550 \times 600$ meters, with the top side 700 m under the surface. Three uranium ore veins, where the mining process is concentrated, are located in this domain. For both the basic materials (uranium ore and surrounding rocks) and the goaf material filling the volume of the extracted ore, linear elastic behaviour is assumed. The discretization of the domain by a regular structured grid has $124 \times 137 \times 76$ bricks, see Fig. 6.2. This leads to a finite element system of 3 873 264 degrees of freedom.

Mathematical modelling aimed at the assessment of geomechanical effects of mining, e.g. comparison of different mining methods from the point of view of stress changes and possibility of dangerous rockbursts. The immediate task was to model four selected stages of successive mining in the considered domain, represented by a four-step sequence of problems with different material distribution.

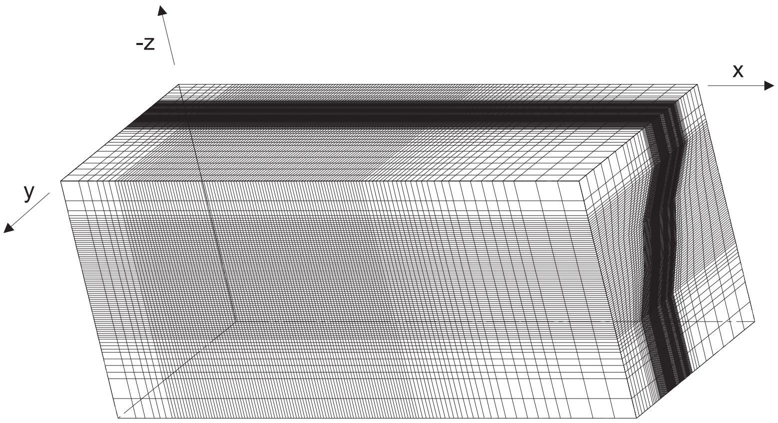


FIG. 6.2. *The DR problem: the mesh*

**6.2. Computing resources.** All the computations mentioned in this study were performed on an entry-level installation of the well-known IBM SP multicomputer, originally equipped with eight POWER2/66.7 MHz (Thin-2) processor nodes with at least 128 MB of system memory. Each node had three network interfaces, Ethernet 10Mbit/s, ATM 155 MBit/s and the proprietary SP High Performance Switch (HPS) 40 MByte/s, allowing performance tests under various communication conditions. Software environment included AIX 4, XL Fortran and Parallel Virtual Machine (PVM) version 3.3.11.

Because of partial hardware and software upgrades, the testing conditions have been changing slightly in the course of time. We tried to keep the consistency of the experiments as much possible, using the processor node above with 128 MB memory as the basic testing platform. The measurements were carried out in non-dedicated mode, but in periods without other (non-system) applications, in case of doubts repeatedly. The best wall-clock times, most important from the practical point of view, are presented.

**6.3. Results.** First, let us show some latest results of the DD-based solver. As this code is still under development and the experiments are going on, we regard them as preliminary.

Tab. 6.1 contains iteration counts and execution times (in seconds) achieved by the PCG-P(DD) when solving the FOOT40 problem with relative accuracy $10^{-3}$. They differ in the application of the coarse grid (see Sect. 4.1). Whereas in the FOOT40 part of the table this technique is not used, in the FOOT40+04 and FOOT40+12 respectively coarse grids $4 \times 4 \times 4$ and $12 \times 12 \times 12$ are applied. In the table, the results are further parametrized by the overlapping factor (OF) and the number of subdomains. The times were obtained for the PVM message passing using the HPS communication subsystem of IBM SP in the IP mode (see above). Subdomain workers as well as the coarse grid worker ran on separate processor nodes, whereas the computationally modest master shared the processor with one of the workers.

As one can observe,

- there is a great variation in the execution time depending on the solution parameters (e.g. the coarse grid size),
- the coarse grid dramatically improves performance,

| *Overlap. fact.* | *OF=0* | | *OF=1* | | *OF=2* | | *OF=3* | | *OF=4* | |
|---|---|---|---|---|---|---|---|---|---|---|
| *Subdomains* | *it.* | *t [s]* | *it.* | *t [s]* | *it.* | *t [s]* | *it.* | *t [s]* | *it.* | *t [s]* |
| FOOT 40 | | | | | | | | | | |
| 2 | 88 | 793 | 69 | 638 | 64 | 611 | 67 | 660 | 67 | 666 |
| 3 | 126 | 778 | 93 | 581 | 87 | 569 | 85 | 573 | 82 | 580 |
| 4 | 133 | 603 | 105 | 505 | 100 | 508 | 98 | 520 | 95 | 527 |
| FOOT 40+04 | | | | | | | | | | |
| 2 | 31 | 293 | 24 | 239 | 24 | 250 | 25 | 260 | 26 | 277 |
| 3 | 40 | 260 | 31 | 204 | 29 | 200 | 29 | 209 | 29 | 220 |
| 4 | 43 | 204 | 34 | 172 | 33 | 179 | 33 | 189 | 33 | 192 |
| FOOT 40+12 | | | | | | | | | | |
| 2 | 17 | 171 | 14 | 186 | 13 | 145 | 14 | 157 | 15 | 170 |
| 3 | 21 | 158 | 16 | 126 | 15 | 121 | 15 | 122 | 15 | 123 |
| 4 | 21 | 108 | **17** | **90** | 17 | 99 | 17 | 102 | 17 | 103 |

TABLE 6.1
*Results of PCG-P(DD) on FOOT40*

- the execution time decreases with the number of parallel tasks (the code is scalable).

Tab. 6.2 compares the best result from the Tab. 6.1 with the performance of the other solvers under consideration on the FOOT40 problem. In addition to the run times, relative speedup ($S_r$) and efficiency ($E_r$) with regard to the sequential version were calculated. The gains of the parallelization are pregnant especially in the PCG-P(DD) case, but recall that careful tuning of the run-time parameters of the parallel solution was necessary.

| *Solver* | *Time [s]* | *Iterations* | *Processors* | *Speedup* | *Efficiency* |
|---|---|---|---|---|---|
| PCG-S | 420 | 49 | 1 | – | – |
| PCG-P(DiD-IF) | 223 | 49 | 3 | 1.88 | 0.63 |
| PCG-P(DiD-VP) | 202 | 8 | 3 | 2.08 | 0.69 |
| PCG-P(DD) | 90 | 17 | 5 | 4.47 | 0.93 |

TABLE 6.2
*Comparison of the solvers on FOOT40*

In the second part of this section we shall concentrate on the DiD-based parallel solvers. Fig. 6.3 shows the interpolated solution times for the PCG-P(DiD-IF) (*dashed*) and PCG-P(DiD-VP) (*solid*) solvers on a sequence of the FOOT benchmarks, ranging from FOOT40 (192 000 equations) to FOOT105 (3 472 875 equations) on Ethernet (*left*) and HPS (*right*) networks. The times for the DR problem (*plus* and *x-mark*, cf. below) are also added as reference.

The graphs document the advantage of the variable preconditioning especially with linear systems of more than 1 million equations. This is the critical size when in our computing environment the DiD based parallel solvers begin to be retarded by the disk operations, since the data does not fit into the main memory. PCG-P(DiD-VP) shows greater data locality and due to the reduction of the expensive outer iterations it lowers the communication/synchronization overhead. This aspect is even more apparent on slow networks like Ethernet.

This observation has been confirmed by the solution of the DR problem. As shown in [4], the procedure was not straightforward, since the task specification lead

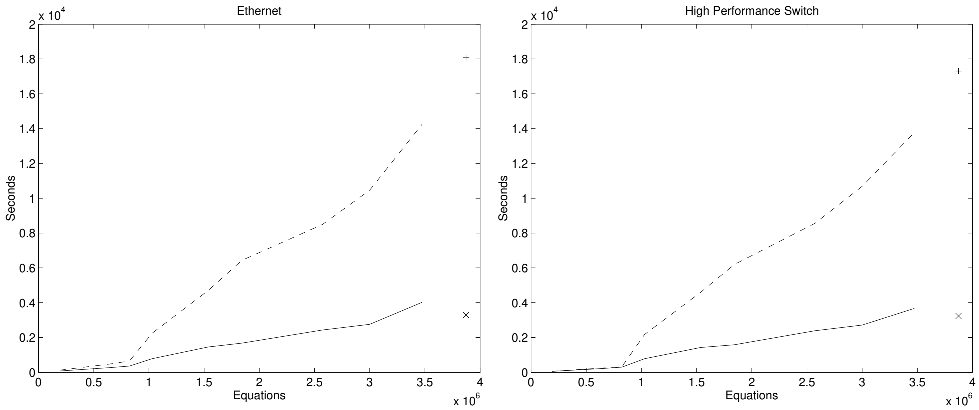


FIG. 6.3. *PCG-P(DiD-IF) and PCG-P(DiD-VP) timings for the FOOT40 – FOOT105 sequence on Ethernet and HPS (details in the text)*

| *Solver\Step* | *St. 0* | *St. 1* | *St. 2* | *St. 3* | *St. 4* | *Total/it.* | *Speedup* |
|---|---|---|---|---|---|---|---|
| PCG-S | 25:45 | 01:03 | 00:36 | 00:36 | 01:02 | 28:17/193 | – |
| PCG-P(DiD-IF) | 12:00 | 00:29 | 00:16 | 00:15 | 00:30 | 13:30/193 | 2.1 |
| PCG-P(DiD-VP) | 02:12 | 00:19 | 00:12 | 00:12 | 00:14 | 03:09/19 | **9.0** |

TABLE 6.3
*Solution times for the DR problem, in hours:minutes*

to four pure Neumann boundary problems, which required some additional mathematical treatment and corresponding modification of the codes, to be able to cope with slightly inconsistent systems. In its final stage, the modelling process comprised five computational steps, each one solving a linear system of nearly four million unknowns. Their duration for the DiD-based solvers[1] is presented in Tab. 6.3.

The solution od the DR problem clearly exposes one less reported beneficial aspect of parallelization: Better hardware utilization due to partitioning to subproblems more appropriate in size, leading to a *superlinear speedup*. Although perhaps not so interesting from the theoretical point of view, it is extraordinary important for practical computations.

**7. Conclusions.** In our contribution, we presented two approaches to the parallelization of the PCG method: the displacement decomposition and the domain decomposition. The way how the preconditioning is realized (IF, VP, various Schwarz preconditioners) as well as the choice of the parallel interface (PVM, MPI) further multiply the possibilities how to design the computer code. In the framework of our in-house GEM32 system we developed in this way a library of FE solvers appropriate for large scale mathematical modelling in various conditions. Whereas the DiD direction seems to resolved, the DD approach deserves further investigations and experiments. In particular, we intend to combine DiD and DD methods to gain even more efficiency and scalability.

**Acknowledgments.** Supporting contracts: Grant Agency of the Czech Republic No. 109/99/1229, Czech Ministry of Education No. LB98273, LB98212 and VŠB – TU Ostrava CEZ:J17/98:2724019.

[1]DR has not been solved by a DD solver yet.

REFERENCES

[1] O. Axelsson and I. Gustafsson, *Iterative Methods for the solution of the Navier equations of elasticity*, Computer Methods in Applied Mechanics and Engineering, 15 (1978), pp. 241–258.

[2] R. Blaheta *Displacement decomposition — incomplete factorization preconditioning techniques for linear elasticity problems*, Numerical Linear Algebra with Applications, 1 (1994), pp. 107–128.

[3] R. Blaheta and R. Kohut, *PCG–1 iterative solver*, Institute of Geonics Cz. Acad. Sci., Ostrava, CR, 1995.

[4] R. Blaheta, R. et al., *Iterative displacement decomposition solvers for HPC in geomechanics*, Proc. Large Scale Scientific Computing LSSC'99 (S. Margenov, P. Vassilevski, P. Zalamov, eds.), Vieweg, to appear.

[5] R. Blaheta, *CG with nonlinear variable preconditioning*, technical report, Institute of Geonics Cz. Acad. Sci., Ostrava, 1999.

[6] R. Blaheta, *Parallel iterative methods*, lecture notes, VŠB – Technical University, Ostrava, 2000.

[7] J. Dongarra, I. S. Duff, D. C. Sorensen and H. van der Vorst, *Numerical linear algebra for high-performance computers*, SIAM, Philadelphia, 1998.

[8] A. Geist et al., *PVM: Parallel Virtual Machine — Users' guide and tutorial for networked parallel computing*, MIT Press, Cambridge, 1994.

[9] B. Smith, P. Bjørstad and W. Gropp, *Domain decomposition: Parallel multilevel methods for elliptic partial differential equations*, Cambridge University Press, New York, 1996.