A FAST POISSON SOLVER FOR TURBULENCE SIMULATION ON PARALLEL COMPUTERS USING SPARSE GRIDS

M. GRIEBEL, W. HUBER, C. ZENGER

Institut für Informatik, Technische Universität München
Arcisstrasse 21, D-8000 München 2
Germany

Abstract. In codes for the simulation of non compressible turbulence a 3D Poisson problem has to be solved for the pressure correction at each time step. This is very CPU-time consuming. Even if fast elliptic solvers like FFT-based methods are applied this task still takes about 40-50% of the overall computation time.

We present so called sparse grid methods which can overcome this problem. In contrast to the usual full regular grid a sparse grid contains substantially less grid points. The accuracy of the obtained solution is nearly the same as for the full grid solution, provided that the solution is sufficiently smooth. Additionally, this approach is well suited for parallelization.

We discuss the ability of the sparse grid Poisson solver in the context of non-smooth right hand sides that occur typically in turbulence simulation. Furthermore we report performance figures for the implementation of the combination technique on different parallel computers and a workstation network.

1. Introduction. In the following section we give a short outlook of the mathematical model used in the FLOWSI code. For details see [FRI90]. Compressed turbulent flows can be described mathematicly by the so called 'homogeneous density flow model'

$$\varrho_{,t} + \varrho u_{j,j} = 0 \tag{1}$$

$$\varrho(u_{i,t} + u_j u_{i,j}) = -p_{,i} + (2\mu s_{ij} - \frac{2}{3}\mu s_{kk}\delta_{ij})_{,j} \tag{2}$$

$$s_{ij} = 0.5(u_{i,j} + u_{j,i}). \tag{3}$$

In FLOWSI the term $\frac{2}{3}\mu s_{kk}\delta_{ij}$ is still be ignored. The discretisation of the equations (1) and (2) is obtained in the code FLOWSI by Chorin's projection technique with an explicit time-step-method (Leapfrog) by

$$u_i^* = u_i^{n-1} + 2\Delta t(C_i^n + D_i^{n-1}) \tag{4}$$

$$u_i^{n+1} = u_i^* - 2\Delta t\delta_i p^{n+1} \tag{5}$$

$$\delta_i u_i^{n+1} = -K^{n+1} \tag{6}$$

where C_i denotes the convection terms, D_i denotes the diffusion terms and K^{n+1} express the known compression rate at time n+1. Furthermore, δ_i describes the central differences.

For the correction of the approximation u_i^* in equation (5) we need the unknown pressure field p^{n+1} from the poisson-equation

$$\delta_i\delta_i p^{n+1} = \frac{\delta_i u_i^* + K^{n+1}}{2\Delta t}. \tag{7}$$

Presently the 3D Poisson-problem is reduced by an FFT to a sequence of 2D Poisson problems. Each is solved by a fast and fully vectoriezed elliptic solver with cyclic reduction. For statistical evaluations more than 100 realizations of the non compressible problem with different starting values have to be solved.

To reduce the overall computation time we intend to replace the existing Poisson-solver (FFT and the elliptic solver) by a new method, the so called combination technique (see [GSZ90]) with is based on the sparse grid approach and hierarchical basis idea. With this method only $O(h_n^{-1}(\mathrm{ld}(h_n^{-1}))^{d-1})$ grid points are needed for solving an elliptic partial differential equation instead of $O(h_n^{-d})$ grid points for d-dimensional problems. Here, $h_n = 2^{-n}$ denotes the mesh size. The accuracy of the sparse grid combination solution with respect to the L_2-Norm or the $L_\infty-$norm is of the order $O(h_n^2(\mathrm{ld}(h_n^{-1}))^{d-1})$ assuming that the solution is sufficiently smooth. With respcet to the energy-norm the solution is $O(h)$ (see [BUNG92] for further details). This is only slightly worse than $O(h_n^2)$ obtained for the usual full grid solution.

The natural coarse grain parallelism of the combination method makes it prefectly suited for MIMD parallel computers and distributed processing on workstation networks. $O((\mathrm{ld}(h_n^{-1}))^{d-1})$ problems of the size $O(h_n^{-1})$ have to be solved independently and can therefore be computed fully in parallel.

For the 3-dimensional case, we report our first results of numerical experiments obtained on the nCUBE2 with 128 processors, the Transputer SuperCluster with 256 transputers, the iPSC/860 with 16 Processors and on a network of 110 HP720-workstations.

2. The combination method. We consider a partial differential equation

$$Lu = f$$

in the unit cube $\bar{\Omega} = [0,1]^d \subset \mathbb{R}^d$ with a linear, elliptic operator L of second order and appropriate boundary conditions. For reasons of simplicity, first, we restrict ourselves to the case $d = 2$ and consider the case $d = 3$ later.

The usual approach is to discretize the problem by a finite element, finite difference method or finite volume on an equidistant grid $\Omega_{n,n}$ with grid size $h_n = 2^{-n}$ in x- and y-direction and to solve the arising linear system of equations $L_{n,n}u_{n,n} = f_{n,n}$. Then, we get a solution $u_{n,n}$ with error $e_{n,n} = u - u_{n,n} = O(h_n^2)$ if u is sufficiently smooth. Here, we assume that $u_{n,n}$ represents an appropriate interpolant defined by the values of the discrete solutions on grid $\Omega_{n,n}$.

For the solution of the discrete system, one of the most effective techniques is the multigrid method. Roughly speaking, the number of operations is of the order $O(h_n^{-2})$ and is therefore proportional to the number of grid points.

Extending this standard approach we now study *linear combinations* of discrete solutions of the problem on different rectangular grids. Let $\Omega_{i,j}$ be the uniform grid on Ω with mesh sizes $h_1 = 2^{-i}$ and $h_2 = 2^{-j}$ in x- and y-direction.

To this end, we consider the so-called combination solution

$$u_{n,n}^c = \sum_{i+j=n+1} u_{i,j} - \sum_{i+j=n} u_{i,j} \tag{8}$$

that has been introduced in [GSZ90]. Here, i ranges from 1 to n. This method is illustrated in Fig. 1.

FIG. 1. *The linear combination of grids $\Omega_{i,j}$ and solutions $u_{i,j}$ with i+j=n+1 and i+j=n, n=4.*

Thus, we have to solve n different problems $L_{i,j}u_{i,j} = f_{i,j}, i+j = n+1$, each with about 2^n unknowns and $n-1$ different problems $L_{i,j}u_{i,j} = f_{i,j}, i+j = n$, each with about 2^{n-1} unknowns and combine their bilinearly interpolated solutions. This gives a solution defined on the so-called sparse grid $\Omega^s_{n,n}$, see Fig. 2. The sparse grid $\Omega^s_{n,n}$ is a subset of the *associated full grid* $\Omega_{n,n}$.

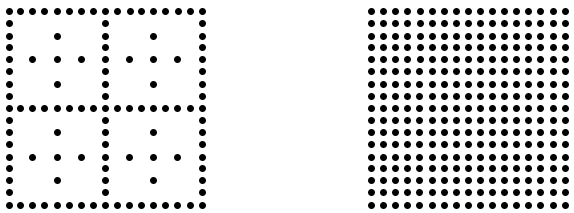

FIG. 2. *The sparse grid $\Omega^s_{4,4}$ and the associated full grid $\Omega_{4,4}$.*

Properties of the combination method. Altogether, the combination method involves $O(h_n^{-1}\mathrm{ld}(h_n^{-1}))$ unknowns in contrast to $O(h_n^{-2})$ unknowns for the conventional full grid approach. Additionally, the combination solution $u^c_{n,n}$ is nearly as accurate as the standard solution $u_{n,n}$. It can be proved (see [GSZ90]) that the error satisfies

$$e^c_{n,n} = u - u^c_{n,n} = O(h_n^2 \mathrm{ld}(h_n^{-1})).$$

This is only slightly worse than for the associated full grid where the error is of the order $O(h_n^2)$. For the proof we assume that u is sufficiently smooth, so that for $u_{i,j}$ (interpolated from grid $\Omega_{i,j}$ to the domain Ω) an *error splitting* of the form

$$u_{i,j} = u + h_i^2\omega_1(h_i) + h_j^2\omega_2(h_j) + h_i^2h_j^2\omega_3(h_i,h_j) \tag{9}$$

holds for any fixed point $(x,y) \in \Omega$ where ω_1, ω_2, ω_3 are smaller than a constant C. If we insert the error splitting formula (9) into equation (8), we see that the leading error terms cancel. We get the estimation

$$|u - u^c_{n,n}| \leq C \cdot \ h_n^2 \cdot (1 + 5/4 \ \mathrm{ld}(h_n^{-1})) = O(h_n^2 \mathrm{ld}(h_n^{-1})).$$

Related sparse grid techniques have been studied by [BUN92], where it is shown that the energy-norm of the error is of the order $O(h_n^{-1})$ which is the same as for the full grid approach.

Three dimensional combination. The combination technique can be extended to the three-dimensional case. We get

$$u^c_{n,n,n} = \sum_{i+j+k=n+2} u_{i,j,k} - 2 \sum_{i+j+k=n+1} u_{i,j,k} + \sum_{i+j+k=n} u_{i,j,k}. \tag{10}$$

This means that all problems *in total* only have $O(h_n^{-1}(\mathrm{ld}(h_n^{-1}))^2)$ instead of $O(h_n^{-3})$ unknowns. Analogous to the two dimensional case, the accuracy deteriorates slightly pointwise (with respect to the L_2- and L_∞-norm) from $O(h_n^2)$ to $O(h_n^2(\mathrm{ld}(h_n^{-1}))^2)$. For details of the proof see Griebel, Schneider, and Zenger [**?**]. Now, we have to solve $O((\mathrm{ld}(h_n^{-1})^2)$ different problems where the size of each problem is of the order $O(h_n^{-1})$. The generalization to higher dimensions is straightforward.

Generalizations. The combination technique is not restricted to the unit square. We have successfully treated problems on distorted quadrilaterals, triangles, and more general domains with polygonal boundaries. Additionally, problems with a non-linear operator and PDE-systems like the Stokes' equations have been solved.

To some extent, the combination method even works in the case of non-smooth solutions. Now, h_i^2 and h_j^2 in (9) have to be replaced by h_i^α, h_j^β with appropriate α and β, but, because of the properties of the combination technique, the leading error terms still cancel. However, for problems with severe singularities the appropriate combination of adaptively refined grids is recommended.

This method will be used to solve the Poisson-problem in the pressure correction step. We remark that all the different discrete problems whose solutions have to be combined are totally independent of each other and can be solved *fully in parallel.*

3. Parallelization aspects of the combination method. For the combination technique in two dimensions, n different problems each with about 2^n unknowns and $n-1$ problems each with about 2^{n-1} unknowns can be solved fully in parallel.

In the 3-dimensional case, $n \cdot (n+1)/2$ problems with about 2^n unknowns, $(n-1) \cdot n/2$ problems with about 2^{n-1} unknowns, and $(n-2) \cdot (n-1)/2$ problems with about 2^{n-2} unknowns can be solved fully in parallel. The situation for more than three dimensions is analogous. This parallelisation potential of the combination technique can be gained already on a relatively coarse grain level.

Additionally, the subproblem solver (e.g. multigrid) can be parallelized. This, however, requires a comparatively fine grain parallelization. In addition to the natural coarse grain parallelism this can only be exploited on massively parallel systems with thousands of processors. Alternatively, the subproblems are ideal for vectorization (see [**?**], [**?**]).

Note that the problem size and the number of different problems that can be treated in parallel is only dependent on the discretization parameter n.

Parallelization concept. The independence of the subproblems can be accommodated by a simple parallel computation model. We propose a farm-like master and slave

structure for the coarse grain parallelization. This is illustrated in Fig. 3 by a p-ary tree with depth one. The master distributes the subproblems to be solved to its slaves and collects the results when they all have terminated.

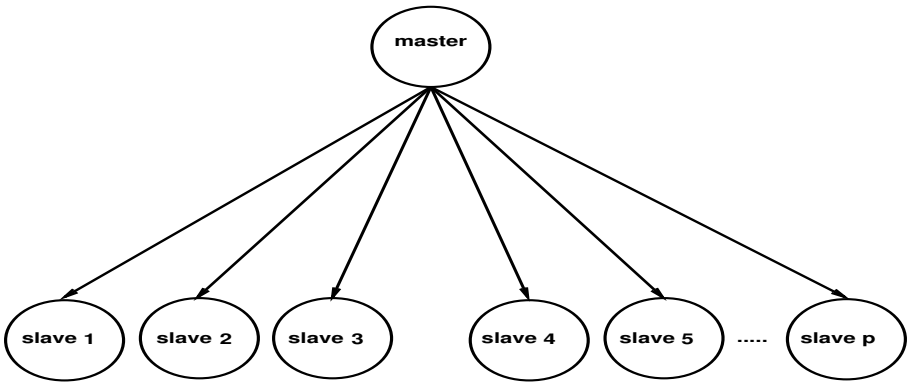

FIG. 3. *master-slave concept*

Regarding the higher dimensional case the parallelization tree remains at depth one and only its breadth grows as the number of subproblems to be solved increases. The subproblem size remains basically the same. This simple structure is suitable because usually the communication is very small compared to the work assigned to each processor. In extreme cases with many parallel tasks, we can avoid the bottlenecks in a $1 \leftrightarrow p$ communication by implementing the logical structure of Fig. 3 with additional *sub-masters* each taking responsibility for a certain number of slaves. This approach can be applied recursively.

For reasons of high efficiency we used in our codes on the nCUBE2 and on the TransputerCluster two slaves for each submaster (so we get a binary-tree). On the workstation-net with 110 HP 720 we used 10 submasters ans associated 10 slaves for each of them. The reason why we do not apply a binary-tree like on the nCUBE2 or the Transputer relies on the structure of the workstation-network.

Load balancing. The different sizes of the different problem classes suggest the following simple load balancing strategy. Assume that p processors are available. First, the n problems of size 2^n are distributed among the processors. We denote the work to solve one of these problems by W. In the case of a multigrid solver, we may assume $W = O(2^n)$. Now, the load of two processors differs at most by W. For two dimensions we are left with $n-1$ problems of size $O(2^{n-1})$ each corresponding to a load of $W/2$. As shown in Fig. 4, two of these problems are used to fill the processors having less load than the others. If, after this step, not all tasks have been assigned to processors, the remaining problems are again evenly distributed among the processors. For the three-dimensional case this algorithm is continued in the canonical way with the problems of size $O(2^{n-2})$.

Following this simple strategy, we achieve that the load of two different processors differs at most by $W/2$ for the 2D case and $W/4$ for the 3D case, provided that the number of problems is sufficiently large with respect to the number of processors. In the 3-dimensional case, the largest efficiency with the highest speed up for a given n is obtained by using

$$P = \text{int}\left[\frac{n(n+1)}{2} + \frac{(n-1)n}{4} + \frac{(n-2)(n-1)}{8}\right] \tag{11}$$

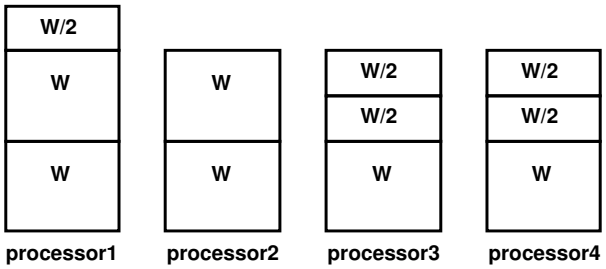

FIG. 4. *A simple load balancing strategy, 2D case, P=4.*

processors. Note at last that this strategy can be improved somewhat if we exploit the fact that for example the problems on grids $\Omega_{i,j,k}$, $i+j+k=n$ do not possess exactly the same number of unknowns and, that thus, the work to solve these problems is not exactly the same.

4. The combination technique for pressure correction . In the previous section we described the combination method and its paralisation properties. We remarked that the combination technique works well for sufficiently smooth solutions (see [GRI90]). Now, we like to see, if the combination method still gives sufficiently accurate solutions for the Poisson-problem which arrise in the pressure correction step for turbulence simulation problems. There, the arrising right hand side for the Poisson-problem are typically not smooth but locally highly oscillating. Consequently, the solution is nonsmooth either but oscillating. To find out, how well the combination technique works in this context, it has to incorporated into the turbulence simulation code FLOWSI. This will be done in the near future.

Here, we give the results of our first numerical experiments which indicate that the combination method might work sufficiently well in turbulence simulation. To this end, we studie the turbulence flow through a channel as a modell problem. Figure 5 shows a 2D-cut through a typical 3D right hand side. We see that the oscillations are large near the wall and small or nonexisting in the interior of the flow.

Figure 6 (left side) shows the oscillated solution $u_{n,n}$ which was achived by solving the Poisson-problem on the $full$ grid. We see, that the solution is non-smooth near the wall. Alternativly, we computed a sparse grid solution $u^c_{n,n}$ by the combination technique and interpolated it to the full grid. This is shown on the right side of figure 6.

Note that the right hand side for the subproblems arising in the combination technique are formed correctly from the full grid right side $f_{n,n}$ by a numerical integration process.

Clearly we see the difference between both solutions. However, the question is if thus difference is substantial enough to outrule the combination solution.

In the finite element or finite volume discretisation process the right hand side is gained usually by numerical integration. Alternativelly, the lumping technique is often used. There, no integration is perforfmed to gain a discrete right hand side but the values on the grid points are taken instead. The difference between these two right hand side can be interpreted as integration error, which is usually of the order $O(h^2)$. Figure 7 shows the integrated right hand side (left) and the assoziated full grid solution $u_{n,n}$.

The full grid solution that is obtained for the integrated right hand side is usually accepted to be as accurate as the solution obtained for the lumping-right hand side. The difference $u_{n,n} - \tilde{u}_{n,n}$ of both full grid solution is shown in the left part of figure 8.

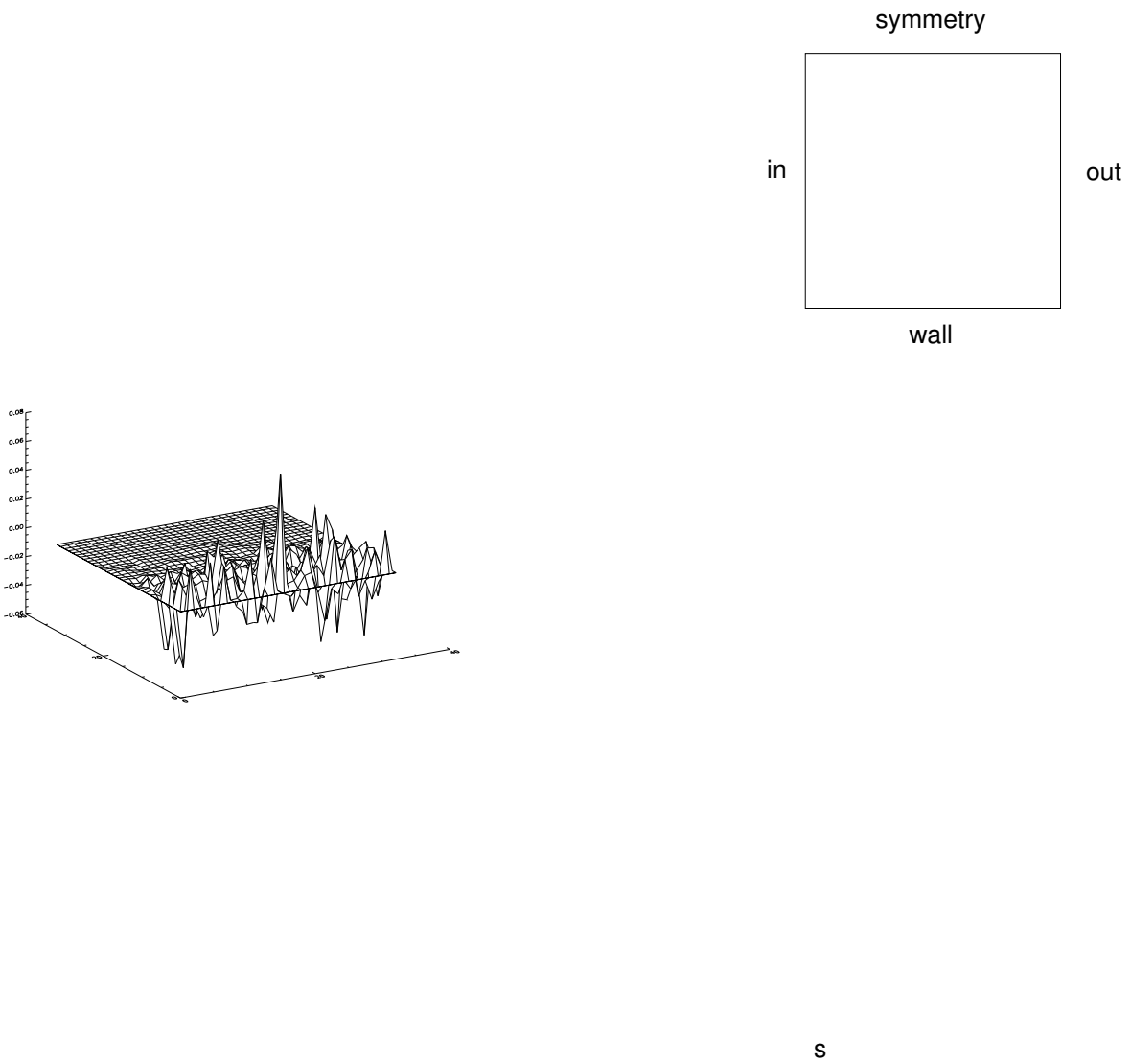

FIG. 5. *Geometry of the modell problem and a typically oszillating right-hand-side*

Additionally, the difference between the combination solution $u^c_{n,n}$ and the solution $\tilde{u}_{n,n}$ is shown in the right side of figure 9.

This shows that, at least for our modell problem the error caused by the combination technique lies within the range of the error caused by the numerical integration error for the right hand side.

Thus, the combination solution can be interpreted as the exact full grid solution for a slightly permutated right hand side, where the permutation is within the integration error range. From this we deduce that the combination method might still be working sufficiently in the turbulence simultion context, because lower frequent errors are eliminated and higher frequent errors are of the same order than the error caused by numerical integration of the right side. Note that we used a 9 point FE stencil for discretisation. This might cause the wiggles seen in the interior region of the right side of figure 8. The use of a 5 point FD-stencil instead should remove the wiggles.

We hope, that the above observations and concluding remarks will be valid also for other turbulence simulation problems then our simple modell problem. This, however, has to be verified in future experiments.

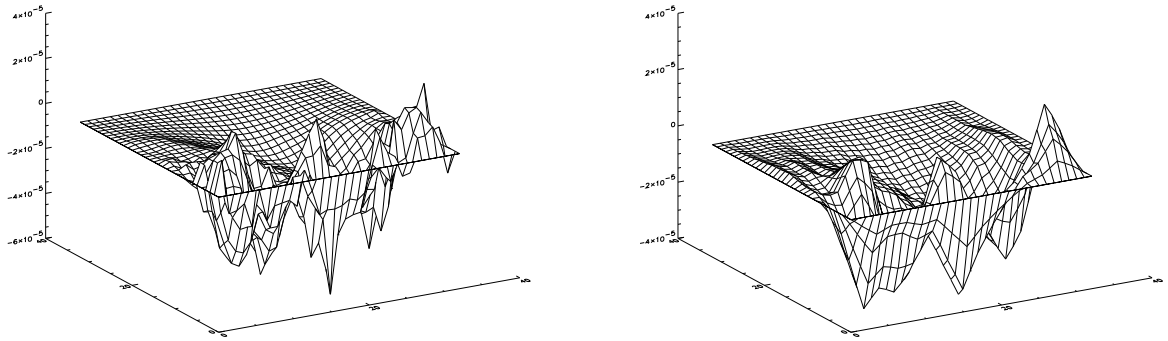

FIG. 6. *The full grid solution $u_{n,n}$ (left) and the combination solution $u^c_{n,n}$ (right) (2D cut)*

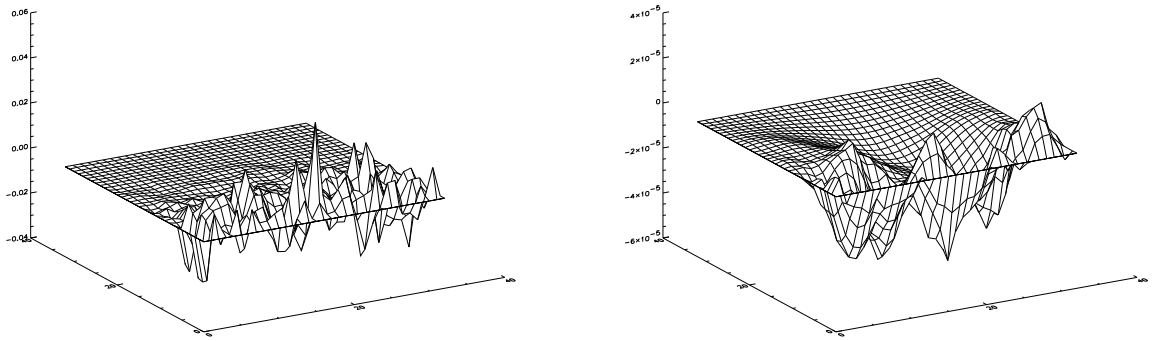

FIG. 7. *Integrated rihgt hand side and assoziated full grid solution* $\tilde{u}_{n,n}$

In the following we consider the parallel potential of the combination method in more details.

5. Numerical experiments on parallel computers. In this section we turn to the results of our numerical experiments. We consider the simple three-dimensional model problem

$$\Delta u = 0 \quad \text{in } \Omega = (0,1) \times (0,1) \times (0,1) \tag{12}$$

with the exact solution

$$u = \sin(\pi x) \cdot \sin(\pi y) \cdot \sinh(\sqrt{2}\pi z)/\sinh(\sqrt{2}\pi)$$

and Dirichlet boundary conditions on $\partial\bar{\Omega}$.

For the solution of the subproblems we use 10 V-cycles of a multigrid method with one step of eight-color Gauss-Seidel pre-smoothing and one post-smoothing. Thus, the computational cost of the parallel tasks is of the same order as the cost of the communication of each solution to the master task. In this sense, the simple Laplace problem is a quite hard test example for the parallel performance of our algorithm. Experiments with more complicated PDE's and PDE-systems like the Stokes' equations gave analogous results.

Note that our code uses full 27-point stencils that are derived from the assembly of trilinear quadrilateral finite elements. The most straightforward finite difference discretization with symmetric second order differences would result in a 7-point stencil, only.

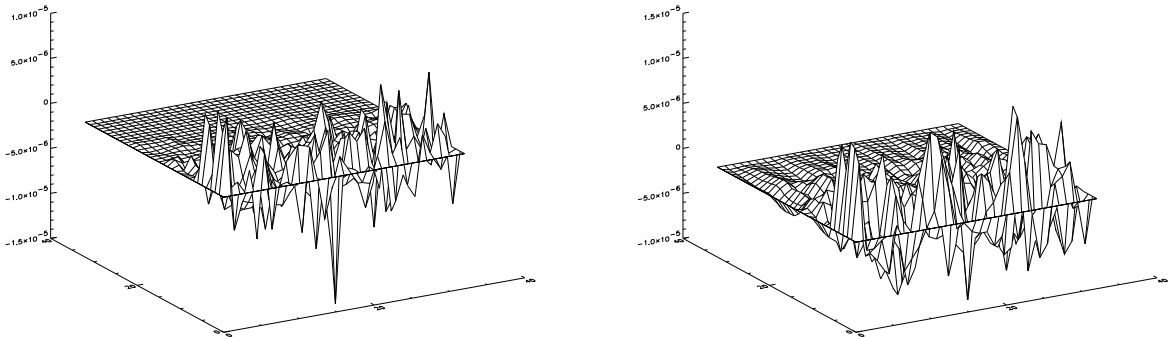

FIG. 8. *Difference* $u_{n,n}$-$\tilde{u}_{n,n}$ *(left) and* $u^c_{n,n} - \tilde{u}_{n,n}$

Furthermore, the stencil is assembled explicitly at each grid point, simulating the solution of a more general, second order elliptic operator with variable coefficients. Table 1 summarizes the number of floating point operations needed by the combination method for different values of n. Here and in the following, the entries of each column have to be multiplied by the associated factor in the second row.

TABLE 1
Flop for combination algorithm on $\Omega^s_{n,n}$.

n	3	4	5	6	7	8	9	10	11	12	13	14
×	10^5	10^6	10^7	10^7	10^8	10^8	10^8	10^9	10^9	10^{10}	10^{10}	10^{10}
Flop	5.449	2.723	1.076	3.720	1.172	3.459	9.610	2.575	6.661	1.676	4.127	9.982

We clearly see the $O(h_n^{-1}(\mathrm{ld}(h_n^{-1}))^2)$-behavior of the three-dimensional combination algorithm.

Experiments on the nCUBE2. Now, we turn to the results which had been achieved on a nCUBE2 with 128 processors. The nCUBE2 is a distributed memory processor system where the processing units are connected in hypercube topology. Each CPU runs at 20 MHz and is rated at 2.4 MFlops double precision. Each processor of our system has 4 MByte main memory.

With this configuration we measured the times shown in table 2.

TABLE 2
Times (in sec.) for combination algorithm on $\Omega^s_{n,n}$ *on the nCUBE2 with P processors.*

$P\backslash n$	3	4	5	6	7	8	9	10	11	12	13
×	10^0	10^0	10^0	10^1	10^1	10^2	10^2	10^3	10^3	10^3	10^4
1	2.9	9.1	31.7	10.5	32.6	9.56	26.4	7.06	18.2	45.8	11.3
2	2.87	5.31	16.9	5.38	16.4	4.8	13.3	3.55	9.17	22.9	5.65
4	1.89	3.52	9.77	3.00	9.24	2.76	7.67	2.02	5.27	13.2	3.22
8	1.77	3.90	7.08	1.66	5.03	1.52	3.96	1.05	2.69	6.71	1.64
16	1.67	2.20	4.11	1.02	2.75	0.78	2.14	0.59	1.43	3.56	0.90
32	1.60	2.26	3.50	0.66	1.67	0.41	1.21	0.30	0.77	1.87	0.45
64	-	-	-	-	-	0.27	0.71	0.18	0.41	1.00	0.27
128	-	-	-	-	-	-	-	-	0.27	0.56	0.14

The first row of table 2 ($P = 1$) displays the run-time of the sequential combination algorithm, so that we see once more that the typical $O(h_n^{-1}(\mathrm{ld}(h_n^{-1}))^2)$-behavior is dominant.

TABLE 3
MFlop per second for combination algorithm on $\Omega^s_{n,n}$ on the nCUBE2 with P processors.

$P \backslash n$	3	4	5	6	7	8	9	10	11	12	13
1	0.19	0.30	0.34	0.35	0.36	0.36	0.36	0.36	0.37	0.37	0.37
2	0.19	0.51	0.64	0.69	0.72	0.72	0.72	0.72	0.73	0.73	0.73
4	0.28	0.77	1.10	1.24	1.27	1.26	1.25	1.27	1.26	1.27	1.28
8	0.31	0.70	1.52	2.25	2.33	2.28	2.42	2.45	2.48	2.50	2.51
16	0.33	1.24	2.62	3.71	4.26	4.44	4.49	4.33	4.66	4.71	4.61
32	0.34	1.21	3.07	5.65	7.03	8.51	7.97	8.46	8.65	8.97	9.16
64	-	-	-	-	-	13.0	13.5	14.5	16.3	16.6	-
128	-	-	-	-	-	-	-	-	24.7	29.9	29.5

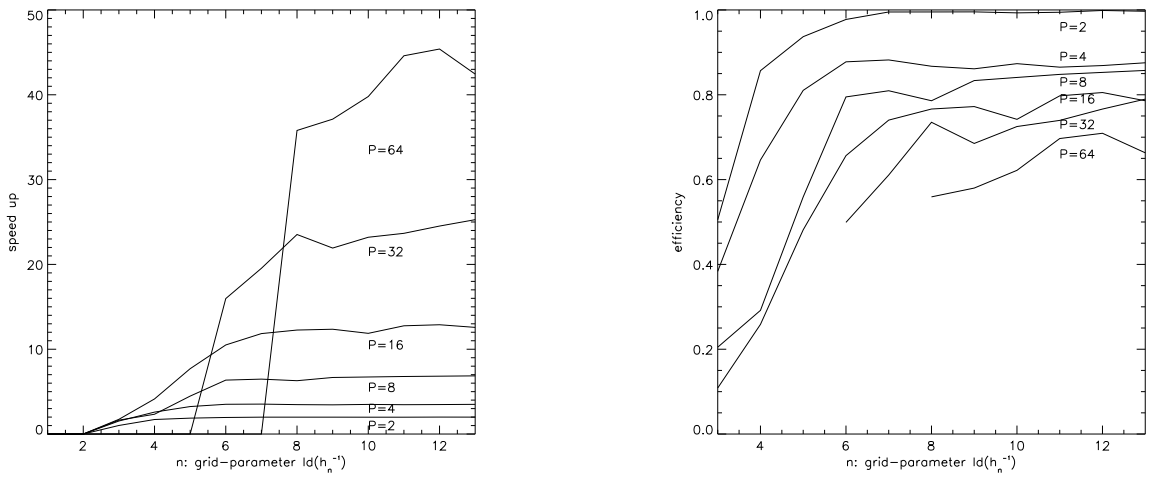

FIG. 9. *Speed up and efficiency on the nCUBE2 for different P*

Table 3 shows the speed in MFlop/second.

The achieved speedup and the resulting efficiency for different problem sizes and processor numbers is displays in fig.9. Clearly, parallelization is inefficient, when the problem is very small. However, when the problem size grows, efficiency and speedup quickly reach a maximal rate. Generally, the efficiency for problems of reasonable size is above 60 percent and is therefore quite satisfactory.

Experiments on the Transputer SuperCluster . The following results have been performed on the Transputer SuperCluster from Parsytec with 256 node running under Helios. Each node has 4 MByte memory and a peek performance of 2.2 MFlops and a practically performance of 0.26 MFlops. In table 4 we show the measured time. Similar to the results on the nCUBE2 the first row of table 4 shows the time for the sequential version. The MFLOP-rate is displayed on table 5.

Figure 10 shows the speedup and efficiency for different numbers of transputers.

Experiments on the iPSC/860 . The results desribed below are made on an intel iPSC/860 with 16 i860 RISC processors Each of them has 8 MByte memory and a practically performance of 1.93 MFlops. The iPSC is a distributed memory processor system with hypercube topology. With the different numbers of prozessors we messured the times shown in table 6. The first row shows the run-time of the sequential algorithm. Table 7 displays the numbers of MFlop per second.

Fig. 11 shows the resulting speed-up and efficency of the iPSC/860.

TABLE 4
Times (in sec.) for combination algorithm on $\Omega^s_{n,n}$ on the SuperCluster with P processors.

$P\backslash n$	4	5	6	7	8	9	10	11	12	13	14
$\times$	10^0	10^0	10^1	10^1	10^1	10^1	10^2	10^2	10^2	10^3	10^3
1	11.09	42.86	14.63	45.73	134.38	372.58	99.72	257.77	648.95	159.92	367.84
2	5.64	21.89	7.38	22.89	67.48	186.70	50.23	129.79	325.60	80.35	194.85
4	3.03	11.67	3.98	12.71	38.25	107.41	28.45	74.24	186.27	42.73	110.94
8	1.76	6.06	2.07	6.83	20.84	55.39	14.74	37.79	94.86	23.20	56.71
16	0.95	3.57	1.26	3.64	10.54	29.07	8.08	19.21	49.02	12.39	30.24
32	0.95	2.69	0.68	2.07	5.29	16.10	4.10	10.59	25.70	6.17	15.44
64	0.95	2.69	0.66	1.47	3.30	8.99	2.40	5.62	13.79	3.39	8.28
128	0.95	2.69	0.66	1.47	3.30	7.92	1.67	3.46	7.29	1.89	4.75
256	-	-	-	-	-	-	-	-	-	-	3.21

TABLE 5
Mflop per second for combination algorithm on $\Omega^s_{n,n}$ on the SuperCluster with P processors.

$P\backslash n$	4	5	6	7	8	9	10	11	12	13	14
1	0.25	0.25	0.25	.026	0.26	0.26	0.26	0.26	0.26	0.26	-
2	0.48	0.49	0.50	0.51	0.51	0.52	0.51	0.51	0.51	0.51	0.51
4	0.90	0.92	0.94	0.92	0.90	0.90	0.91	0.90	0.90	0.97	0.90
8	1.55	1.78	1.80	1.72	1.66	1.73	1.75	1.76	1.77	1.78	1.76
16	2.87	3.01	2.95	3.22	3.28	3.31	3.19	3.47	3.42	3.33	3.30
32	2.87	4.00	5.50	5.66	6.53	5.97	6.28	6.29	6.52	6.69	6.46
64	2.87	4.00	5.66	7.99	10.49	10.69	10.74	11.85	12.15	12.18	12.06
128	0.89	2.43	3.82	6.76	9.53	12.10	15.45	19.28	23.00	21.83	21.03
256	-	-	-	-	-	-	-	-	-	-	30.41

Experiments on a network of high performance workstations. The following experiments have been performed on a network of 110 HP720 workstations. The workstations are organized in 10 clusters. Each cluster consists of a disc-server and 10 discless clients, each with 16MByte main memory. It is noteworthy that the total memory capacity of this system is more than 1.6 GByte and thus larger than most machines presently considered as supercomputers. The HP720 workstation is based on HP's Precision RISC-architecture. This architecture achieves a peak-rate of 17 MFlop per second, so that the peak rate of the network is 1.84 GFlop per second.

Our experiments indicate that for practical applications a careful and sophisticated optimization is required to obtain at least about 50% of the single processor peak rate. Our present code, that has not been specially optimized yet, runs at 5-8 MFlops on a single processor.

Of course, the application of the network as a parallel supercomputing system is

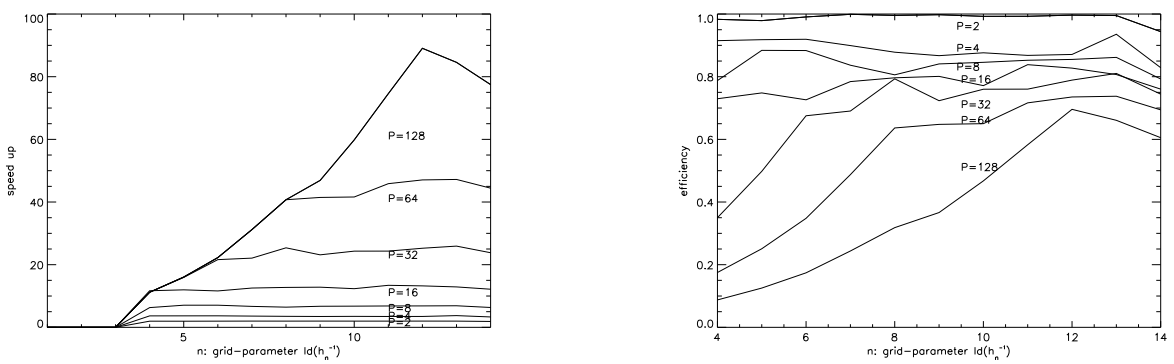

FIG. 10. *Speed up and efficiency on the SuperCluster SC-256 for different P.*

TABLE 6
Times (in sec.) for combination algorithm on $\Omega^s_{n,n}$ on the iPSC/860 with P processors.

$P\backslash n$	4	5	6	7	8	9	10	11	12	13	14
$\times$	10^0	10^0	10^0	10^0	10^1	10^1	10^2	10^2	10^2	10^3	10^3
1	2.2	6.9	22.1	65.7	18.9	51.4	13.6	34.8	86.9	21.3	51.47
2	1.3	3.7	11.1	33.6	9.6	25.7	6.9	17.6	43.5	10.7	25.7
4	0.9	2.2	6.2	18.6	5.4	14.8	3.9	10.0	24.9	6.1	14.6
8	0.7	1.5	3.4	9.9	3.1	7.7	2.0	5.1	12.7	3.1	7.5
16	0.6	1.0	2.2	6.4	1.6	4.1	1.1	2.7	6.6	1.7	4.0

TABLE 7
MFlop per second for combination algorithm on $\Omega^s_{n,n}$ on the iPSC/860 with P processors.

$P\backslash n$	4	5	6	7	8	9	10	11	12	13	14
1	1.26	1.56	1.68	1.78	1.83	1.87	1.89	1.91	1.93	1.93	1.94
2	2.08	2.89	3.36	3.49	3.61	3.73	3.74	3.79	3.86	3.87	3.88
4	2.96	4.79	5.95	6.30	6.43	6.47	6.66	6.66	6.74	6.81	6.83
8	4.00	7.30	10.86	11.80	11.29	12.41	12.76	13.01	13.17	13.42	13.39
16	4.42	10.91	17.26	18.30	21.95	23.50	23.65	24.95	25.41	24.26	25.05

additionally handicapped by several more constraints. The ethernet connection is very slow compared to the processing speed of a single workstation. Furthermore, it is not possible to use the workstations as a dedicated system. Even at night hours when no regular users are present, it is practically impossible to control the effect of background processes on the elapsed time and performance. For each of the parallel tasks we compute the sum of the user and system time and neglect idle times caused by virtual memory effects, disc-IO and background processes. The maximum of these times plus the execution time of the master process are used as the overall time.

Fortunately, the communication demands of our application are so simple that a quite naive implementation already gives quite satisfactory results. A shell script is used to start the execution of the slave processes. This is done in two levels. The top-master starts cluster-master processes on the cluster servers. The cluster-masters then distribute the slave processes on the cluster-clients. The cluster servers are left idle while the slaves are in operation. Each master waits for the termination of its sub-masters or slaves, respectively, which stores the computed result in an associated file. When all slaves have terminated, the master collects the results from the files to calculate the final solution. This relies on the network file system (NFS).

With this implementation we measured times that are shown in Table 8. Table 9

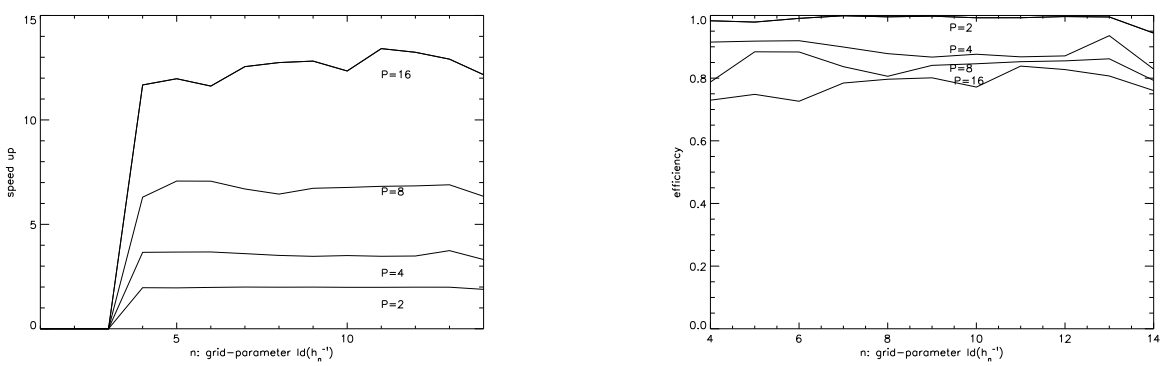

FIG. 11. *Speed up and efficiency on the iPSC/860 for different P.*

shows the achieved MFlop per second.

TABLE 8
Times (in sec.) for combination algorithm on $\Omega^s_{n,n}$ on network with P HP720-workstations.

$P\backslash n$	4	5	6	7	8	9	10	11	12	13	14
1	0.54	2.03	6.85	20.94	60.87	168.17	448.07	1156.19	2929.23	7225.50	18176.33
2	0.29	1.03	3.45	10.49	30.45	84.17	224.04	581.35	1456.82	3644.77	9118.20
4	0.17	0.58	1.90	5.92	17.10	49.11	129.82	332.61	833.64	2041.04	5101.98
8	0.11	0.29	0.97	3.07	9.38	24.77	65.70	168.65	421.60	1038.61	2611.60
16	0.09	0.20	0.56	1.86	4.70	12.99	37.15	85.45	216.15	553.39	1385.15
32	0.09	0.17	0.33	0.98	2.39	7.21	18.34	46.99	113.69	272.61	707.92
64	0.24	0.33	0.55	0.87	1.72	4.21	11.01	24.95	61.54	152.23	380.82
100	0.14	0.24	0.33	0.76	1.63	2.63	8.91	17.75	44.86	101.0	237.53

TABLE 9
MFlop per second for combination algorithm on $\Omega^s_{n,n}$ on network with P HP720-workstations.

$P\backslash n$	4	5	6	7	8	9	10	11	12	13	14
1	6.25	6.86	7.51	8.01	8.41	5.70	7.87	5.76	7.98	5.71	5.49
2	9.39	10.45	10.75	11.17	11.36	11.42	11.49	11.46	11.51	11.32	10.95
4	16.02	18.55	19.58	19.80	20.23	19.57	19.83	20.03	20.11	20.22	19.57
8	24.76	37.11	38.35	38.17	36.88	38.80	39.18	39.50	39.76	39.73	38.22
16	30.26	53.80	66.43	63.00	73.60	73.98	69.29	77.95	77.55	74.57	72.07
32	30.26	63.30	112.73	119.58	144.74	133.29	140.35	141.75	147.44	151.34	141.01
64	30.26	63.30	112.73	122.07	201.11	228.27	233.79	266.97	272.38	271.08	262.12
100	19.45	44.84	112.73	71.90	212.22	275.36	324.74	386.37	401.02	409.27	420.25

On 100 processors we achieved a rate of more than 420 MFlops which is a quite impressive result. On a CRAY YMP/8128 with 4 processors, a comparable algorithm (see [**?**]) ran with about the same MFlops rate.

Fig.11 shows the resulting speed up and efficiency for varying problem sizes and processor numbers. Once more, an efficiency of more than 60 percent can be seen.

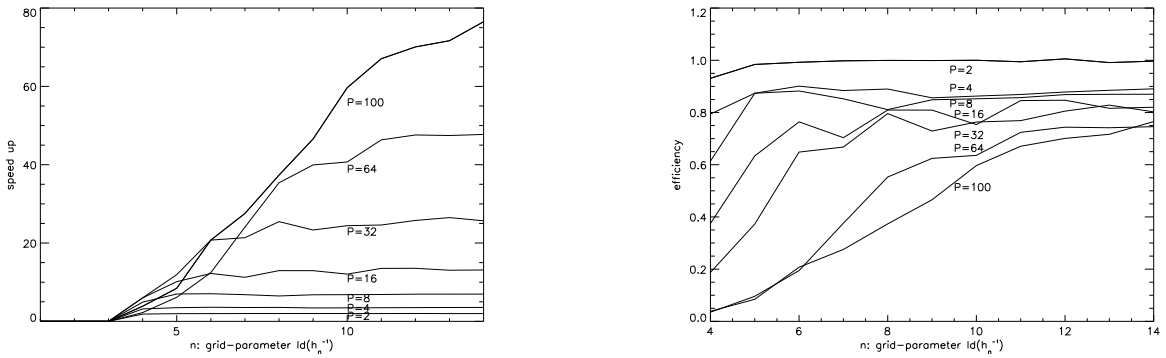

FIG. 12. *Speed up and efficiency on the workstation network for different P.*

6. Concluding remarks. In this report we present the combination technique for two and three dimensions. In contrast to the usual approach which needs in the 2-D case $O(h_n^2)$ grid points, our combination technique method works with only $O(h_n^{-1} ld(h_n))$ grid points, where h_n denotes the employed grid size. The accuracy of the solution is quite the same, providing that the solution is sufficiently smooth. Additionally, the new method is perfectly suited for a so called coarse grain parallization. We verify these properties on various distributed memory computers and on a workstation network. Furthermore, we gave the results of first experiments which indicates that the combination method

might work sufficiently also for turbulence simulation problems. In the near future we will incorpurate an improved version of our parallel 3D code into FLOWSI.