PARALLEL DOMAIN DECOMPOSITION AND ITERATIVE REFINEMENT ALGORITHMS *

PETTER E. BJØRSTAD , RANDI MOE AND MORTEN SKOGEN

Summary. Algorithms for the solution of partial differential equations based on a subdivision of the spatial domain, has received much interest in recent years. To a large extent this has been motivated by the new generation of parallel computers. This algorithmic approach can introduce independent parallel tasks of variable granularity, depending on the subdivision and can therefore be adapted to a wide range of parallel computers. We review some of the progress that has been made and report on numerical experiments that illustrate the convergence behavior. We also describe parallel implementations on both shared memory computers (Alliant FX/8) and on local memory systems (Intel iPSC/2 and network connected workstations).

1. Introduction. Algorithms for the solution of partial differential equations based on a subdivision of the spatial domain, has received much interest in recent years; cf. e.g. Glowinski et al, [12], and Chan et al, [7]. This is to a large extent due to their potential on multiprocessor computing systems.

The general technique of domain splitting is not new. H. A. Schwarz introduced such a technique in a theoretical context more than hundred years ago [21]. Engineers, in particular within the field of structural analysis, have used the principle extensively, partly to ease the task of modeling complex structures, partly in order to organize large computational tasks on computers with limited memory [2, 20]. The subdomain solution technique has traditionally been Gaussian elimination and the global solution has been obtained by completing what can be viewed as a block Gaussian elimination process.

A related problem concerns the solution of a composite problem having one or more levels of local grid refinement. In this case we assume that each refinement patch is a regular grid as opposed to fully adaptive schemes where the refinement can be restricted to individual elements. We call these methods iterative refinement algorithms.

We assume that there exists an algorithm for solving the problem on a given subdomain, our main task is to study the computational process of obtaining the global solution using a multiprocessor system. This iterative process typically involves the solution of each subproblem one or two times in every iteration. The iteration should ideally converge in a fixed, very small number of steps, independent of the number of subdomains and of the grid size employed in each subdomain. We consider each subdomain solve as a separate computational task. Our main interest is to study the computational efficiency of this approach on different parallel computing systems. A production code should also simultaneously consider the finer granularity parallelism inside each subdomain solve, this may enhance parallel efficiency and reduce the load balancing problems. If the complexity of the subdomain solver is worse than linear in the number of unknowns, then the overall complexity is reduced as we "divide and conquer" the domain into more subdomains. We note, that this is indeed often the case in complex, industrial problems. The programming effort needed to implement near optimal algorithms may not be justified even when

* Institutt for Informatikk, University of Bergen, Høyteknologisenteret, N-5020 Bergen, Norway. This work was supported in part by the Norwegian Research Council for Science and the Humanities under grant D.01.08.054 and by The Royal Norwegian Council for Scientific and Industrial Research under grant IT2.28.28484.

such algorithms are well known. The potential gain is even larger when the individual subdomains can be processed in parallel.

We emphasize that both domain decomposition algorithms and iterative grid refinement methods may be incorporated into existing large scale industrial codes without the need for a complete restructure of the code. In this setting, the subproblem solver is typically the original code itself. The introduction of subproblems can reduce the complexity of the geometric representation or make the data structure better suited to the given computer configuration. In the case of refinement, one should again be able to employ the original solver restricted to the refined patch, this approach is quite different from the elegant, adaptive refinement strategy found in the code PLTMG by Bank [1], the cost is the introduction of more unknowns than strictly necessary, the potential saving lies in the possibly simpler data structure, ease in vectorization and the potential use of fast, specialized solvers. These methods are perhaps most attractive precisely in the context of three dimensional problems and nontrivial industrial applications.

2. A brief theoretical description of the algorithms. We give a very brief outline of the fundamentals of the algorithms to be tested. For a more thorough derivation of the theory and the convergence properties of these methods, we refer to [3, 9, 16, 17, 24]. To simplify the presentation, we assume that the elliptic operator is the Laplacian and that we have a zero Dirichlet condition. Thus,

$$\begin{aligned} -\Delta u &= f \qquad in\ \Omega, \\ u &= 0 \qquad on\ \partial\Omega. \end{aligned} \tag{1}$$

The region Ω is bounded, two- or three-dimensional, with a Lipschitz continuous boundary. We use a variational formulation of the problem and seek a solution $u \in H_0^1(\Omega)$, such that

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx = f(v) , \qquad \forall\, v \in H_0^1(\Omega) .$$

Both the theory and the computer algorithms depend on the inner product defined by $a(\cdot,\cdot)$. Thus in this paper orthogonality and symmetry always refer to this inner product.

The problem is discretized by finite elements in the customary fashion; cf. Ciarlet [8] and we define our discrete problem and the corresponding approximate solution $u_h \in V^h$ by

$$a(u_h, v_h) = f(v_h) , \qquad \forall\, v_h \in V^h. \tag{2}$$

In domain decomposition algorithms we subdivide Ω into N subdomains $\Omega^i, i = 1, \ldots, N$, we say that Ω^i overlap with Ω^j if their intersection is nonempty. Associated with each subdomain Ω^i we define the space $V_i = V^h \cap H_0^1(\Omega^i)$.

In the analysis of grid refinement algorithms we consider a nested set of domains $\Omega^i \subset \Omega^j,\ i > j$, where we assume that Ω^i has been refined relative to the triangulation of Ω^j. We again associate a space $V_i = V^h \cap H_0^1(\Omega^i)$ with each Ω^i. It is most simple to consider the problem in terms of hierarchical basis functions, see Yserentant [25]. Such a representation provides a direct sum of subspaces V_i (i.e. no overlap), while the nodal representation gives rise to a subspace overlap corresponding to the coarse mesh functions also being represented in the nodal basis of the refined mesh.

3. Multiplicative Algorithms. Let us consider the Schwarz algorithm in its classical form. There are N fractional steps. Let u^n be the n-th iterate. Then the updated solution u^{n+1} is determined by the successive solution of N subproblems for $i = 1, 2, \ldots, N$

$$\begin{aligned} -\Delta u^{n+i/N} &= f && \textit{in } \Omega^i, \\ u^{n+i/N} &= 0 && \textit{on } \partial\Omega^i \cap \partial\Omega, \\ u^{n+i/N} &= u^{n+(i-1)/N} && \textit{on } \partial\Omega^i \setminus (\partial\Omega^i \cap \partial\Omega). \end{aligned}$$

Note that the solution can be defined globally after each fractional step, by extending the solution on Ω^i to the entire domain by using the most recently computed values. Following P. L. Lions [14, 15], we transform these equations into variational form,

$$a(u_h^{n+i/N} - u_h^{n+(i-1)/N}, v_h) = a(u_h - u_h^{n+(i-1)/N}, v_h)\ , \qquad \forall\, v_h \in V^h \cap H_0^1(\Omega^i).$$

Here u_h is the exact solution of equation (2) and $u_h^{n+i/N} - u_h^{n+(i-1)/N} \in H_0^1(\Omega^i)$ since the boundary values are constant from one fractional step to the next. Following Lions, these equations can be restated in terms of orthogonal projections,

$$u_h^{n+i/N} - u_h^{n+(i-1)/N} \quad = \quad P_i(u_h - u_h^{n+(i-1)/N})\ ,$$

where $P_i\ ,\ i = 1, 2 \ldots N$, is the orthogonal projection of V^h on $V_i = V^h \cap H_0^1(\Omega^i)$.

If we denote the contribution from the subdomain Ω^i, (by subassembly) to the global stiffness matrix K by $K^{(i)}$, and permute the unknowns appropriately, we see that the projection P_i corresponds to the mapping

$$y = P_i x = \begin{pmatrix} K^{(i)^{-1}} & 0 \\ 0 & 0 \end{pmatrix} Kx.$$

The error $e_h^n = u_h^n - u_h$ satisfies the relationship

$$e_h^{n+1} = \prod_{i=0}^{N-1} P_{N-i}^{\perp} e_h^n\ , \tag{3}$$

where $P_j^{\perp} = I - P_j$ is the orthogonal complement of P_j. Schwarz' method can therefore be viewed as a straightforward iterative method for solving the equation

$$(I - \prod_{i=0}^{N-1} P_{N-i}^{\perp}) u_h = g_m \tag{4}$$

for a special right hand side g_m. The operator is not symmetric, but it can be made symmetric by introducing more partial steps traversing the subdomains in reverse order back to Ω^1. This symmetric version can be accelerated, for example with the conjugate gradient method.

In the case of iterative refinement, the FAC method [17, 18, 19] can be interpreted as a Schwarz' method applied to the subspaces V_i with overlap $V_i \cap V_j$, $i \neq j$. Let $u_h^n \in V^h$ be the current approximation and assume we have N levels of refinement. The FAC algorithm can be stated as follows:

Algorithm 1 (FAC).
for $i = 1, \ldots, N$
compute the solution $u_i \in V_i(\Omega^i)$ *of*

$$a(u_h^n + u_i, v_i) = f(v_i) \;, \qquad \forall\, v_i \in V_i(\Omega^i)$$

and set $u_h^n \leftarrow u_h^n + u_i$.

Again, one can write similar expressions to (3) and (4) when introducing the appropriate projection operators. The form of these equations, being polynomials in the projections, motivate the term multiplicative algorithms. These algorithms are not perfect for parallelization since the polynomial terms of degree higher than one, implies sequential operations on subdomains. It should be noted that the polynomial degree will be quite low in many practical situations, because only terms corresponding to nontrivial overlap situations will enter. Thus, even the multiplicative algorithm may provide a considerable number of independent tasks suitable for a multiprocessor system.

4. Additive algorithms. In order to achieve fast convergence, the resulting overall algorithm can often be viewed as an accelerated iterative method combined with a good preconditioning. Preconditioners often take the form of approximate solvers. Many important approximate solvers can be viewed as solving subproblems that can be formulated as

Find $u_i \in V_i$ such that

$$a(u_i, v_i) = f(v_i) \;, \qquad \forall\, v_i \in V_i, \tag{5}$$

for a number of subspaces $V_i \subset V^h$ such that

$$V^h = V_1 + V_2 + \ \ldots\ + V_N.$$

In general, $V_i \cap V_j \neq \{0\}$. An approximate solution $u_h^n \in V^h$ to

$$a(u_h, v_h) = f(v_h) \;, \qquad \forall\, v_h \in V^h \tag{6}$$

is then taken to be $u_h^n \approx \sum_{i=1}^{N} u_i$. We notice that u_i in (5) is nothing but the orthogonal projection of $u_h \in V^h$ on the subspace V_i, i.e. $u_i = P_{V_i} u_h$. This approach corresponds to what we shall call additive algorithms, the approximate solution can be computed in parallel, and the individual contributions need only be added together at the end of every major iteration.

The additive algorithm can be thought of as a block Jacobi type method. In the multiplicative algorithms the information obtained in one subproblem is used as data when solving later subproblems. This method may therefore be viewed as a Gauss Seidel type algorithm.

It should be noted that grid refinement algorithms like the ones introduced in [13, 18, 19] also can be described using this framework. Let $u_h^n \in V^h$ be the current approximation and assume we have N levels of refinement. One possible AFAC algorithm can be described as follows:

ALGORITHM 2 (AFAC).

1. **for** $i = 1, \ldots, N$

compute the solution $u_i \in V_i(\Omega^i)$ *of*

$$a(u_h^n + u_i, v_i) = f(v_i), \qquad \forall\, v_i \in V_i(\Omega^i).$$

for $i = 1, \ldots, N\text{-}1$

compute the solution $w_i \in V_i(\Omega^{i+1}) \cap V_{i+1}(\Omega^{i+1})$ *of*

$$a(u_h^n + w_i, v_i) = f(v_i), \qquad \forall\, v_i \in V_i(\Omega^{i+1}) \cap V_{i+1}(\Omega^{i+1}).$$

Set $w_N = 0$.

2. Set $u_h^{n+1} \leftarrow u_h^n + \sum_{i=1}^{N} (u_i - w_i)$.

A different motivation for the additive algorithms can be obtained from the discussion of multiplicative methods, by just retaining the linear terms in the polynomial (4). The resulting linear system will then take the form:

$$(\sum_{i=1}^{N} P_i) u_h = g_a. \tag{7}$$

The appropriate right hand side g_a can be computed by observing that

$$a(P_i u_h, v_h) = f(v_h)\ , \qquad \forall\, v_h \in V_i(\Omega^i).$$

Therefore, at the expense of solving one subproblem in each subdomain, the appropriate right hand side can be constructed.

5. Convergence. The analysis of these methods is completely understood in the case of two domains. In [3] the relationship between additive and multiplicative Schwarz' methods is fully analyzed, and it is shown that the spectra of the iteration operators are related in a very precise way. An eigenvalue of the form $1 - \mu_i^2$ in the multiplicative case corresponds to two eigenvalues of the form $1 \pm \mu_i$ in the additive case. The resulting condition number for the additive method is therefore bounded by 4 times the condition number of the multiplicative method. A similar result is given in [4] for the relationship between FAC and AFAC. This is a somewhat pessimistic result since it indicates that we may need twice as many iterations using the additive method. More details on the two-domain Schwarz algorithm can be found in [6].

The multilevel case has been addressed in [10, 16, 18, 19, 24]. These results all show that under reasonable assumptions, the iterative refinement methods are indeed optimal in the sense that they converge in a number of iterations that is independent of both the number of refinement levels and of the mesh spacing. A more detailed investigation of the two and three level case together with careful numerical experiments in the general

$h_1 =$		1/8	1/16	1/32
AFAC	λ_{max}	1.434421	1.436717	1.439223
	$\lambda_{min} + \lambda_{max}$	2.004588	2.000063	2.000005
FAC	μ_{max}	0.428042	0.436631	0.439218

TABLE 1
A 5-level iterative grid refinement.

case, is also given in [4]. That report indicates that the spectra of the iteration operators behave surprisingly similar also in the multilevel case. The eigenvalues for the additive method are only slightly perturbed from the form $1 \pm \mu$ and approach this form as the grid size parameter h tends to zero. Table 1 displays the extreme eigenvalues for a five level FAC and AFAC test example. The domain is a unit square and we refine a quarter of the square repeatedly towards one corner. Each square has a uniform, piecewise linear triangulation with the same number of interior nodes. The grid size parameter of the coarsest grid is denoted by h_1. We note that the eigenvalues behave very similar to the two level theory outlined above. We observe that the condition number of the iteration matrix never exceeds 2.5 in this example. We have never observed μ_{max} exceed a half, but this does clearly depend on the ratio of refined to unrefined area.

In the case of the Schwarz' method applied to the multi-domain case, a recent report by Widlund, [23] derives a recursive bound for the condition number of the resulting symmetrized iterative method of the form:

$$\kappa(\Omega^1, \cdots \Omega^N) \leq \kappa(\Omega^1, \Omega^2 \cup \Omega^3 \cup \cdots \Omega^N) \cdots \kappa(\Omega^{N-1}, \Omega^N). \tag{8}$$

Again, this multiplicative bound may appear pessimistic, but a simple argument shows that as long as no mechanism for global exchange of information is present, the condition number must grow at least as fast as the square of the number of subregions that must be traversed in order to connect any two domains. Numerical experiments carried out in a three domain model case [5] indicate that the bound (8) is indeed fairly sharp in many cases. The condition number is only slightly better than the predicted upper bound when the overlapping regions are quite small. We have in our experiments, never seen the estimate more than 20 percent off. As the overlap is increased, the bound becomes very sharp, and the actual condition number is often within one percent of the bound.

6. Parallel implementations. The additive and multiplicative methods have been tested on two different parallel architectures, a shared memory Alliant FX/8 with eight computational elements and a local memory Intel iPSC/2 with 16 processors. In addition, the methods have been tested on a network of up to 8 Sun 4/60 workstations connected by an Ethernet. In this case message passing primitives were provided by the Cosmic Environment software package [22] from Caltech.

The parallel computation time as a function of the number of processors can be divided into different components:

$$T(p) = T_s + \frac{T_p}{p} + T_0(p) \ ,$$

where T_s denotes the running time of the sequential part of the program, T_p the sequential running time of the part of the program which can be run in parallel, p the number of processors, and $T_0(p)$ the synchronization and communication overhead (see [11]).

On the shared memory machine Alliant FX/8 the overhead function consists of the costs of using critical regions to schedule the tasks. If traversing the critical region takes time t_c, then it takes pt_c time units for all processors to finish one critical region. Hence the overhead associated with m parallel regions is

$$T_{0_S}(p) = m(pt_c - t_c)\ ;$$

the subtraction of t_c is added to insure that $T_{0_S}(1) = 0$. In addition various data access conflicts involving the hierarchical memory system of the Alliant FX/8 will affect the parallel performance.

On a local memory machine based on a message-passing system, the main overhead cost is the cost of communication. Assuming k messages each of length l_i, and that $t(l_i, p)$ is the time needed to send a message of size l_i in a network of p processors, the overhead function is

$$T_{0_L}(p) = \sum_{i=1}^{k} t(l_i, p).$$

The communication network of the Intel iPSC/2 has quite different characteristics from the local network of workstations. The main component of the time needed to send a message between two nodes of the iPSC/2, is the setup time, i.e. it is more economical to send one long message than several short messages. Messages between one pair of nodes may not influence the time needed to send a message between another pair of nodes. The performance of the Ethernet however, is heavily dependent of the total number of packets in the network. A large number of packets will increase the number of collisions which again increase the number of packets sent. The maximum size of a packet is 1500 bytes which means that what is assumed to be a medium sized message on the iPSC/2 will result in several packets on the Ethernet. Hence, in addition to sending as few messages as possible, a major concern should be to keep the size of the messages on the Ethernet as small as possible.

In addition, the network of the iPSC/2 has a maximum bandwidth of 2.3 MB/s while the corresponding characteristic of the Ethernet is 1.25 MB/s. Depending on the size of the messages, we have been able to utilize up to 1.9 MB/s bandwidth on the iPSC/2. Between two workstations we are able to utilize up to 0.7 MB/s bandwidth. The computational units of the network (Sun 4/60) have been measured to be 4-5 times faster than the computational units of the iPSC/2 for a typical subdomain solve. These characteristics result in a ratio between arithmetic costs and communication costs which is more than an order of magnitude worse for the distributed system compared to the iPSC/2.

On a shared memory machine it is easier to explore parallelism of fine granularity than on a local memory machine. The communication costs will easily be too expensive on a local memory machine if the size of the tasks between each synchronization gets too small. This must be taken into consideration when implementing the algorithms on local memory machines, where the goal must be to find an optimal balance between the degree of parallel computation (T_p) and the amount of communication ($T_0(p)$). This issue will be further discussed in sections 8 and 9.

7. Test problems. In our test examples the domain is the unit square and in the case of local refinement, we refine a quarter of the square repeatedly towards one corner

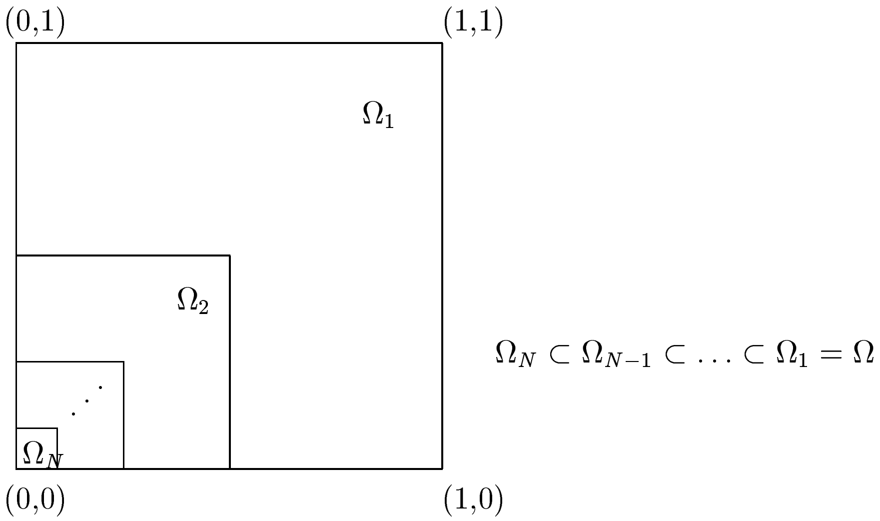

FIG. 1. *The multi-level model case.*

as in figure 1. With only one patch on each level of refinement, this model problem represents in a sense the worst case model. Several independent patches on each level will increase the potential of parallel processing. The domain is decomposed as in figure 2 for the 4-level Schwarz' method. This decomposition indicates the subdomains that can be solved in parallel when using the classical multiplicative Schwarz' method.

For all problems our strategy has been to solve the problems on different subdomains/levels in parallel. As solvers we have used a $O(n^2 \log n)$ fast Poisson solver in the case of iterative refinement and a "slow" Poisson solver in the case of the Schwarz' methods. The "slow" solver is based on doing $O(n)$ one-dimensional Fourier transforms each of length n using the $O(n^2)$ algorithm (SFT). Here n is the number of unknowns in each direction in a subdomain. Our FFT-based Poisson solver can only handle sequences of length n, where $n+1$ has many small factors, efficiently. This is inconvenient for the analysis of the Schwarz' methods, since this method will produce subproblems of different sizes (when the overlap region is included). Each step of our algorithm consists of solving in parallel, problems which differs slightly in size (i.e. the corner, edge and interior subdomains are different because of the boundary), but the computational costs may differ substantially when using such an FFT based fast Poisson solver. The interpretation of our experiments becomes more difficult due to this effect, and we therefore decided to work with a solver that behaves more regularly. In addition, an $O(n^3)$ solver may be more realistic in an industrial setting (Band Gauss would be $O(n^4)$). Even with the "slow" solver we experience efficiency loss due to synchronization. For the multiplicative algorithm there are seven such synchronization steps in each main iteration as opposed to only one for the additive algorithm. Therefore the speedup for the additive Schwarz' method will always be better than for the multiplicative algorithm if the rest of the code is fully parallelized.

Naturally, the parallel part of the algorithm increases as the solver becomes more dominant. The use of a multigrid based Poisson solver, would be the best choice if one wanted to solve the given problem in this context, however, in this paper the given problem serves mainly as an illustration. We discuss this further in section 9, providing examples and comparisons using both the fast Poisson solver and the $O(n^3)$ "slow" algorithm. Finally, we remark that the termination criterion used for all the numerical experiments reported in the following sections, requires the initial residual norm to be reduced by six

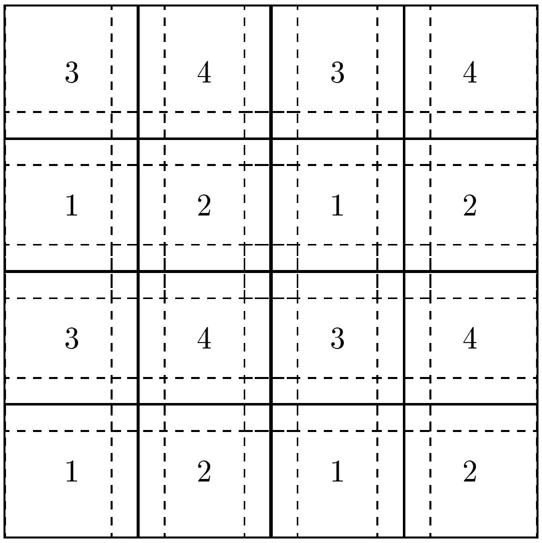

FIG. 2. *Domain decomposition for the 4-level Schwarz' method.*

orders of magnitude.

8. Parallel iterative refinement. For the AFAC method the strategy has been to assign the different levels to different processors. The main task for every processor is to solve the system of equations of the local grid. The work which involves combining the solutions on the different grids and setting up the right hand side of the system of equations for the next iteration, i.e. the interpolation and restriction, is sequential in a global sense. Information have to be transferred from one grid to the others. It is however, possible to compute within each level in parallel, i.e. we are ending up with parallelism of different granularities. This will only cause a small reduction in efficiency on the shared memory machine while as mentioned in section 6, the communication costs will increase substantially on a local memory machine.

In order to avoid fine grained parallelism, an alternative to a sequential exchange of information from the coarsest to the finest level (or vice versa) through all the intermediate levels, is to have an all-to-all communication. This strategy causes the number of messages sent in each iteration to increase from $O(N)$ to $O(N^2)$ where N is the number of levels and, of course, the total number of bytes sent in each iteration also increases substantially. The computational work will increase as well, partly because contributions from all coarser or finer grids have to be added to the current level instead of one contribution from the next coarser or next finer level, and partly because more general algorithms have to be used in order to combine the contributions from different levels with arbitrary grid ratios. The potential gain of this approach on the iPSC/2, is that all the extra computation and communication can be carried out in parallel, and by reducing the idle time of the processors, the overall efficiency can be improved. Preliminary tests showed however, just minimal improvements. Therefore all the reported results on the local memory machines refer to an implementation where the flow of information between the different grids goes through all the intermediate grids (see algorithm 3). Hence there will be an increase of the sequential part of the program as the number of levels and the number of processors increase and a near linear speedup will never be achievable.

With a small number of interior nodes on each grid the amount of work done between each synchronization is limited. Table 3 shows that the speedup achieved on Intel iPSC/2

Alliant FX/8								
	h_1=1/16		h_1=1/32		h_1=1/62		h_1=1/64	
# proc.	sec	speedup	sec	speedup	sec	speedup	sec	speedup
1	6.4		18.7		272.2		55.1	
2	3.4	1.9	9.8	1.9	142.4	1.9	28.5	1.9
4	1.8	3.6	5.1	3.7	74.0	3.7	15.0	3.7
8	1.0	6.4	2.8	6.7	37.9	7.2	8.2	6.7

TABLE 2
A 8-level AFAC with different grid sizes.

Intel iPSC/2								
	h_1=1/16		h_1=1/32		h_1=1/62		h_1=1/64	
# proc.	sec	speedup	sec	speedup	sec	speedup	sec	speedup
1	15.2		67.8		901.7		292.6	
2	8.4	1.8	36.7	1.8	472.9	1.9	157.9	1.9
4	5.0	3.0	20.8	3.3	245.0	3.7	88.0	3.3
8	3.7	4.1	13.2	5.1	131.6	6.9	53.6	5.5

TABLE 3
A 8-level AFAC with different grid sizes.

Distributed system								
	h_1=1/16		h_1=1/32		h_1=1/62		h_1=1/64	
# proc.	sec	speedup	sec	speedup	sec	speedup	sec	speedup
1	4.0		16.5		167.5		63.7	
2	4.3	0.9	10.3	1.6	84.2	2.0	33.9	1.9
4	7.6	0.5	11.0	1.5	48.5	3.5	23.6	2.7
8	14.4	0.3	18.6	0.9	38.0	4.4	25.4	2.5

TABLE 4
A 8-level AFAC with different grid sizes.

increased substantially when h_1 decreased, where h_1 is the grid size parameter of the coarsest grid. On the Alliant FX/8 there is only a small increase in the speedup as h_1 decreases (see table 2). A further increase in the number of unknowns on each level may cause a decreasing speedup depending on the amount of cache memory available on the machine. Similar results for the distributed system is given in table 4. It is obvious that due to the slower communication and faster computations, a certain problem size is needed in order to reduce the total computing time by including more workstations. This is further stated in figure 5 where it is apparent that for the least time consuming test problem (h_1=1/64), increasing the number of processors from 4 to 8 results in an increase in the total computing time. On the other hand, with a less efficient solver (simulated by h_1=1/62), we can still gain speedup by going from 4 to 8 processors. But for all reported problem sizes the efficiency deteriorate rapidly with an increasing number of processors.

In all tables the entry with $h_1 = 1/62$ is included to show the effects of the fast Poisson solver on certain problem sizes, as described in section 7. In this case the solver represents a larger percentage of the total computing time and the importance of the sequential parts are reduced (i.e. T_s is reduced). This leeds naturally to a better speedup.

ALGORITHM 3 (AFAC).

```
RECEIVE initial message from host and set up rhs for grids on this node
repeat
    if (finer grids on other nodes)
        if (restriction message)
            RECEIVE restriction within patch from previous node
        else if (final message)
            SEND solution to host; break
        end if
    end if
    compute restriction
    if (coarser grids on other nodes)
        compute restriction
        SEND restriction to next node
    end if
    for all grids on this node
        solve the system of equations
        if (not finest level)
            solve coarser problem and subtract the solution from the previous solution on this grid
    if (coarser grids on other nodes)
        RECEIVE solution within patch from next node
        interpolate this solution onto coarsest grid on this node
    end if
    for all grids on this node
        interpolate solution from coarse to finer grid
    if (finer grids on other nodes)
        SEND solution within next patch to previous node
    for all grids on this node
        update the solution
        if (next coarser grid on another node)
            SEND updated solution along interface to next node
        if (next finer grid on another node)
            SEND updated solution along interface to previous node
    if (next coarser grid on another node)
        RECEIVE updated solution along interface from next node
    if (next finer grid on another node)
        RECEIVE updated solution from previous node
    compute the residual and let norm of residual propagate
    if (norm of residual < epsilon)
        SEND final messages to all nodes and solution to host
until convergence
```

Each iteration of AFAC consists of N large problems and $N-1$ minor problems where

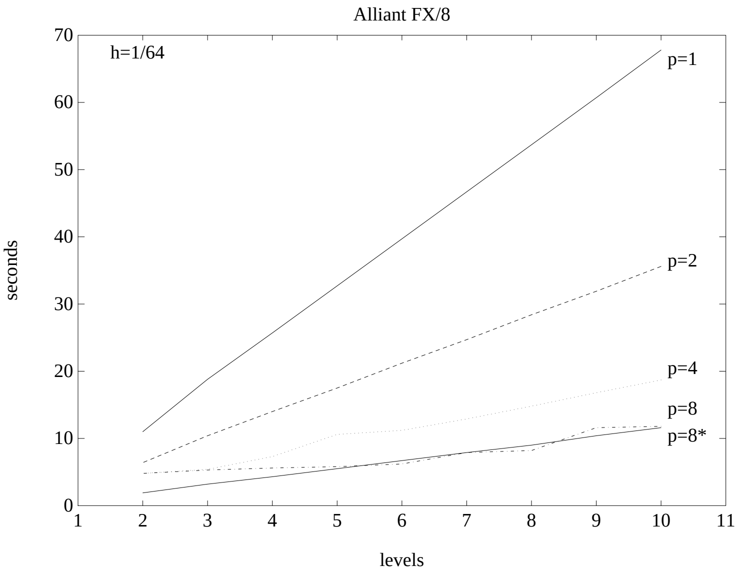

FIG. 3. *AFAC with different levels of refinement but with constant grid size h_1=1/64 where p is the number of processors. The implementation where a parallel solver is used is indicated by p=8*.*

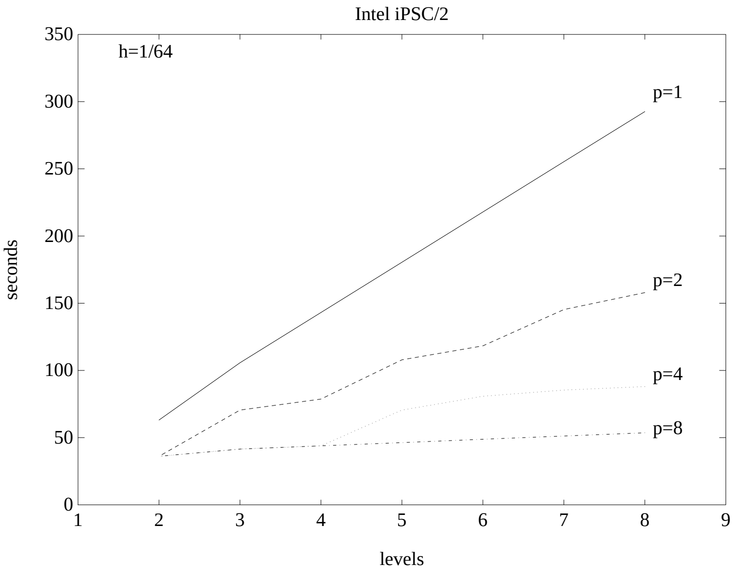

FIG. 4. *AFAC with different levels of refinement but with constant grid size h_1=1/64 where p is the number of processors.*

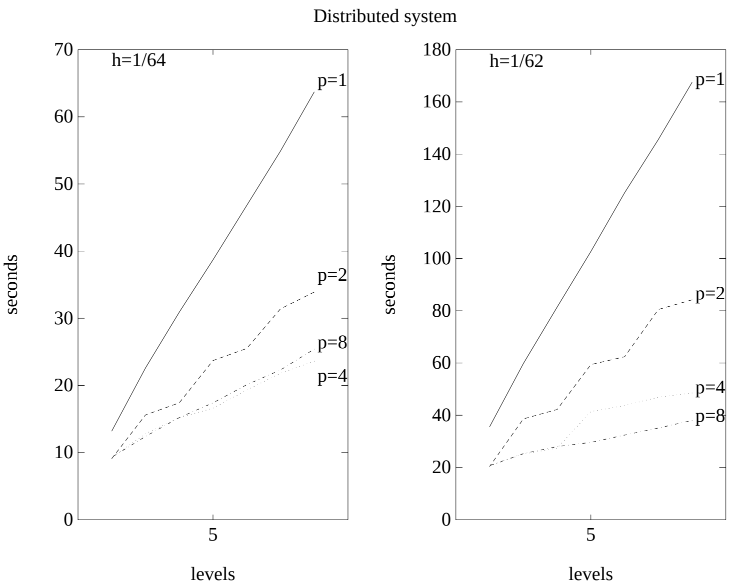

FIG. 5. *AFAC with different levels of refinement but with constant grid size $h_1=1/64$ and $h_1=1/62$ where p is the number of processors.*

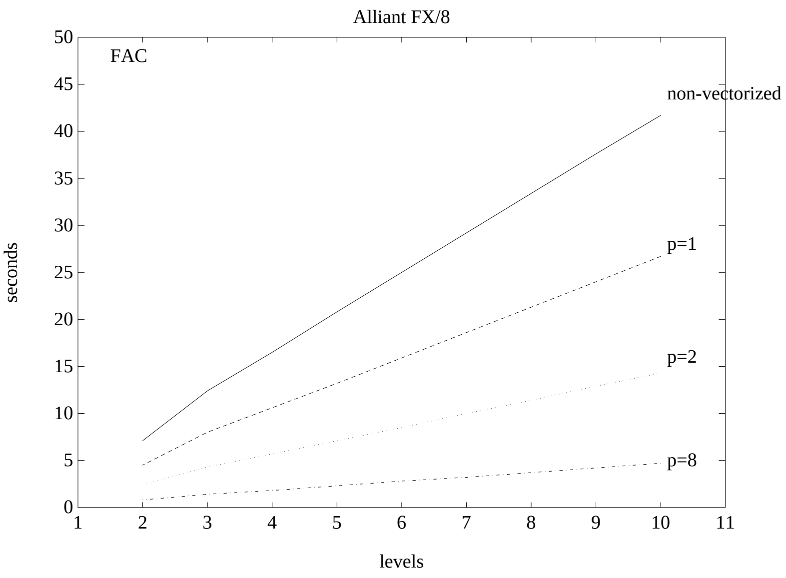

FIG. 6. *FAC with different levels of refinement but with constant grid size $h_1=1/64$ where p is the number of processors. A parallel solver is used on each level.*

	Subdomains = 4		Subdomains = 16		Subdomains = 64	
overlap	Additive	Multipl.	Additive	Multipl.	Additive	Multipl.
3	8.38	1.29	24.1	2.79	88.3	9.13
5	5.87	1.10	14.9	1.94	53.1	5.84
7	4.74	1.03	9.99	1.56	34.6	4.30

TABLE 5

Condition numbers for the additive and the multiplicative Schwarz iteration as a function of the overlap and the number of subdomains.

N is the number of levels (see algorithms 2 and 3). In the current cube implementation the number of levels must be a multiplum of the number of processors to get a reasonably good load balance. These restrictions do not exist on the Alliant FX/8 as the $2N$-1 problems are completely shared among the processors. This effect can clearly be seen in figures 3, 4 and 5.

Figure 3 also shows the results of using a parallel solver on each level instead of distributing the different grids among the processors. With our choice of solver which is very well suited for parallel processing, this variant gave equally good results. The speedup with this strategy is naturally less sensitive to the number of levels. With a parallel solver it is also possible to achieve good speedup with FAC, the multiplicative method. Figure 6 shows the results on the Alliant FX/8.

When evaluating these results one should keep in mind that they are a kind of worst case results because of the special model problem. With more than one patch on each level which would be more likely in practice, the number of independent subtasks will increase and hence the possibilities of exploring parallelism will improve.

9. Parallel Schwarz algorithms. Both the multiplicative and the additive Schwarz algorithms have been accelerated by using the conjugate gradient method. The additive Schwarz' method decomposes almost completely into independent subtasks, i.e. T_s is close to zero, independent of the number of subdomains. The potential slowdown depends on how the number of subdomains map onto the number of processors.

Different tests have been carried out to get estimates of the condition numbers of the multiplicative and additive iteration operators as a function of the overlap and of the number of subdomains. We observe that the condition number is independent of the step size when the number of subdomains is kept fixed and the overlap is kept relatively constant, i.e. the number of overlapping grid points increases with decreasing step size.

As figure 2 shows, all subdomains have an equal overlap into all their neighboring subdomains. We have decided to number the overlap as the total number of common points (in one dimension) between two subdomains. Let A and B be two neighboring subdomains with a common boundary, and let Â and B̂ be the corresponding extensions of A and B by the overlap into the neighboring subdomain. The total size of the overlap is then defined in the following way: The number of common points between Â and B plus the number of common points between A and B̂ plus 1 (the last point being on the boundary between A and B). This means that with an overlap equal to one, only the internal boundary has common grid points.

Table 5 shows the condition numbers of the multiplicative and the additive algorithms for various numbers of subdomains and with variable overlap. The step size h is constant

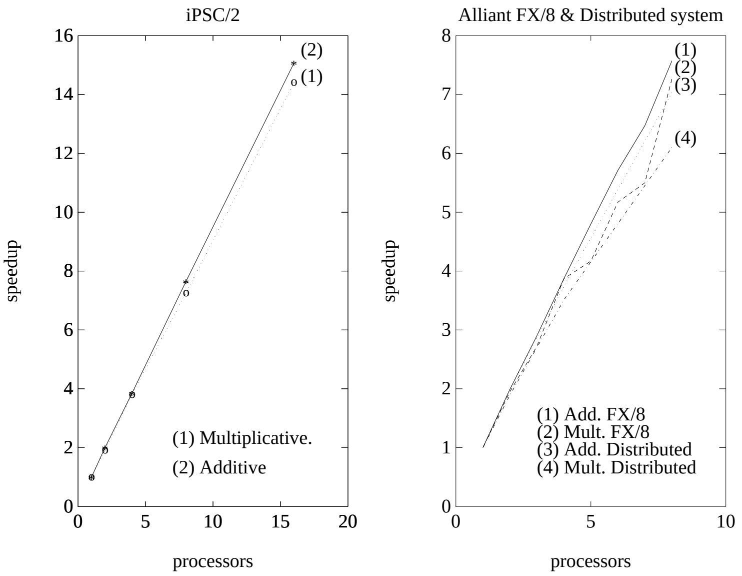

FIG. 7. *Speedup for a testproblem with 64 (8x8) subdomains, h=1/128 and overlap=3.*

($h = 1/64$), and the overlap equals the overlap in the 64 subdomain case (i.e. $7x7$ interior nodes in each subdomain). In the two other cases (4 and 16 subdomains) we have increased the size of the subdomains and their overlap in such a way that their sizes are kept relatively constant. As the table shows, the condition number of the additive algorithm grows very rapidly as the number of subdomains increases.

In figure 7 we have computed the speedup for a test problem for different numbers of processors, (Alliant FX/8, $p = 1, \ldots, 8$, iPSC/2, $p = 2^n, n \leq 4$, Sun 4/60, $p = 2^n, n \leq 3$). From these results it is easy to see the weaknesses of the algorithms. The multiplicative algorithm has a lower speedup than the additive due to the 7 synchronization steps in each main iteration and because of more communication. This have been improved on the shared memory machine by introducing a "pool of tasks". Instead of solving level 1, level 2, etc., we put a label on the subdomains as soon as they are ready to be computed, and the processors pick new domains from the pool of tasks. With this modification we are left with only one synchronization step in each iteration, and the speedup for the multiplicative and the additive algorithm reach the same level. With such a pool of tasks the speedup is no longer dependent on how the number of subdomains on each level map onto the number of processors as we experience with the 7 synchronization steps. It also turns out that the pool algorithm is more independent of the solver. In the previous tests we decided to use a SFT instead of a FFT because of the more regular behavior (se section 7). When most of the synchronization is eliminated, we would expect the pool-algorithm to behave more regularly. This in fact happens, but we also notice that it is superior to the additive algorithm which have the same number of synchronization steps. In table 6 we have given the speedup for an example run on the Alliant FX/8.

From the tests we also experience that this new multiplicative algorithm (in CPU-costs) is slightly better than the additive Schwarz algorithm. However, this will only be true when the number of processors is low compared to the number of subdomains. If the main strategy is to assign one processor to each subdomain and the number of subdomains is N, the additive algorithm will be able to use N processors effectively, while the multiplicative algorithm only can use $N/4$.

	Solver	
Algorithm	Slow Fourier	Fast Fourier
Multiplicative	7.3	5.9
Additive	7.6	6.3
Pool-multi.	7.6	6.8

TABLE 6

Speedup with 8 processors for the three algorithms with two different solvers. The testproblem has 64 subdomains with 9x9 interior nodes in each subdomain, and the overlap consists of one gridpoint only.

	Additive		Global Additive	
overlap	Iterations	Cond. number	Iterations	Cond. number
1	36	409	15	12.5
3	26	195	13	7.33
5	22	124	12	6.02
7	19	88.7	13	5.73
9	17	67.4	13	5.56
11	16	53.2	13	5.48
13	14	42.9	13	5.44
15	13	34.8	13	5.34

TABLE 7

Condition numbers and iteration counts as a function of overlap for test problems with fixed number of subdomains (64) and fixed number of interior grid points (15) in each subdomain.

ALGORITHM 4 (GLOBAL ADDITIVE SCHWARZ ALGORITHM).

repeat
    for (all domains on this node) update residual
    for (all domains on this node)
        compute α_k, add them up
    SEND this sum to all other nodes
    RECEIVE all other $\alpha_k's$, add up to global
    compute β_k
    for (all domains on this node) update p-vector
    compute rhs (Ap) for the preconditioner
        for all domains on this node
            SEND "boundary values" to neighboring domains
            RECEIVE all your "boundary values"
        perform Ap on all domains
    start preconditioning
        for all domains on this node
            solve a Dirichlet problem
        if (coarse problem on this node)
            solve coarse problem
            SEND coarse grid to allnodes
        else
            RECEIVE coarse grid
        end if
        interpolate coarse grid to fine grid
        for all domains on this node
            SEND overlap to all neighboring domains
            RECEIVE overlap from neighboring domains
            add solutions
    end preconditioning
    find global a_k (similar to global α_k)
    for (all domains on this node) update x-vector
until convergence

		Global Additive	
overlap	interior points	Iterations	Cond. number
1	7 by 7	13	7.35
1	15 by 15	15	12.5
1	31 by 31	17	22.5
3	15 by 15	13	7.33
5	15 by 15	12	6.02

TABLE 8

Condition numbers and iteration counts as a function of the overlap and the number of interior grid points for test problems with 64 subdomains.

In [9] Dryja and Widlund introduced a modification for the additive Schwarz' method. A new global problem with the crosspoints between the subdomains as its nodes, should be solved in addition to the subproblems. With this extra solve the condition number becomes independent of the number of subdomains, and the dependency of the overlap is very low. Algorithm 4 describes this global additive Schwarz algorithm on the hypercube. It is also worth noticing that the extra solve can be done in parallel with the others, and as long as the size of the problems are comparable there will be no loss of efficiency. Tests have been done to compare the additive and the global additive Schwarz algorithms. In table 7 we keep the number of subdomains and the number of interior points in each subdomain constant, while the overlap vary. In addition to the condition numbers, we have also given the number of iterations for the two methods. In table 8 we vary both the number of interior points and the overlap for the global additive Schwarz algorithm, and finally in table 9 we keep the number of points constant while we increase the number of subdomains. In the last example the size of the overlap is held relatively constant to the number of interior points in the subdomains.

10. Conclusions. We have implemented and tested both Schwarz type domain decomposition algorithms as well as iterative grid refinement algorithms on several parallel computer architectures. Our results show that these methods quite easily can be tailored to different parallel computers. Our examples are used as illustrations, in production codes there will often be potential for even more parallel tasks, in particular a level of finer granularity inside the solver on each subdomain. Both multiplicative and additive versions of the two algorithms have been tested. Our results indicate that the additive algorithms perform almost as well as the multiplicative, when a parallel computer is used, the ease of parallelization makes the additive algorithms fully competitive. The simultaneous transfer of information between all grids in the iterative refinement algorithms do not seem to improve the parallel performance in such a way that it improves the overall process. With respect to the Schwarz algorithms our results show that using a minimal overlap is most efficient in terms of total work. In this case only the gridpoints on the common interior boundaries between subregions are used as overlap nodes, thus providing a simple picture of the domain splitting.

REFERENCES

[1] R. E. BANK, *PLTMG - user guide*, tech. report, University of California, San Diego, 1988.

	Additive		Global Additive	
subdomains	Iterations	Cond. number	Iterations	Cond. number
4	10	16.2	11	7.07
16	17	51.5	14	7.17
64	27	194	12	7.27
256	47	768	11	7.23
1024	88	3116	10	7.20

TABLE 9

Condition numbers and iteration counts as a function of the number of subdomains. The number of global grid points is fixed (255x255), and the overlap is relative constant. The overlap for 1024 *subdomains is one gridpoint only.*

[2] K. BELL, B. HATLESTAD, O. E. HANSTEEN, AND P. O. ARALDSEN, *NORSAM, a programming system for the finite element method. Users manual, Part 1, General description.*, NTH, Trondheim, 1973.

[3] P. E. BJØRSTAD, *Multiplicative and Additive Schwarz Methods: Convergence in the 2 domain case*, in Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1989, SIAM.

[4] P. E. BJØRSTAD AND R. MOE, *Implementation and convergence of iterative refinement algorithms.*, tech. report, Institute of Informatics, University of Bergen, Thormøhlensgate 55, N - 5008, Bergen, Norway, 1990.

[5] P. E. BJØRSTAD AND M. SKOGEN, *A comparison of multiplicative and additive Schwarz*, tech. report, Institute of Informatics, University of Bergen, Thormøhlensgate 55, N - 5008, Bergen, Norway, 1990.

[6] P. E. BJØRSTAD AND O. B. WIDLUND, *To Overlap or Not to Overlap: A Note on a Domain Decomposition Method for Elliptic Problems*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1053 – 1061.

[7] T. F. CHAN, R. GLOWINSKI, J. PÉRIAUX, AND O. WIDLUND, eds., *Domain Decomposition Methods for Partial Differential Equations II*, Philadelphia, 1989, SIAM. Proceedings of the Second International Symposium on Domain Decomposition Methods , Los Angeles, California , January 14 - 16, 1988.

[8] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, 1978.

[9] M. DRYJA AND O. B. WIDLUND, *An additive variant of the Schwarz alternating method for the case of many subregions*, Tech. Report 339, also Ultracomputer Note 131, Department of Computer Science, Courant Institute, 1987.

[10] ——, *Some domain decomposition algorithms for elliptic problems*, Tech. Report 438 also Ultracomputer Note 155, Department of Computer Science, Courant Institute, 1989. Proceedings of the Conference on Iterative Methods for Large Linear Systems held in Austin, Texas, October 19 - 21, 1988, to celebrate the sixty-fifth birthday of David M. Young, Jr.

[11] H. P. FLATT AND K. KENNEDY, *Performance of parallel processors*, Parallel Computing, (1989), pp. 1–20.

[12] R. GLOWINSKI, G. H. GOLUB, G. A. MEURANT, AND J. PÉRIAUX, eds., *Domain Decomposition Methods for Partial Differential Equations*, Philadelphia, 1988, SIAM. Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Paris, France, January 1987.

[13] L. HART AND S. MCCORMICK, *Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Basic ideas*, Parallel Computing, 12 (1989), pp. 131–144.

[14] P. L. LIONS, *On the Schwarz alternating method. I.*, in First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Philadelphia, 1988, SIAM.

[15] ——, *On the Schwarz alternating method. II.*, in Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1989, SIAM.

[16] J. MANDEL AND S. MCCORMICK, *Iterative solution of elliptic equations with refinement: The model multi-level case*, in Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and

O. Widlund, eds., Philadelphia, 1989, SIAM.

[17] ——, *Iterative solution of elliptic equations with refinement: The two-level case*, in Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1989, SIAM.

[18] S. McCormick, *Fast adaptive composite grid (FAC) methods*, in Defect Correction Methods: Theory and Applications, K. Böhmer and H. J. Stetter, eds., Computing Supplementum 5, Springer-Verlag, Wien, 1984, pp. 115–121.

[19] S. McCormick and J. Thomas, *The fast adaptive composite grid (FAC) method for elliptic equations*, Math. Comp., 46 (1986), pp. 439–456.

[20] J. S. Przemieniecki, *Matrix structural analysis of substructures.*, Am. Inst. Aero. Astro. J., 1 (1963), pp. 138–147.

[21] H. A. Schwarz, *Gesammelte Mathematische Abhandlungen*, vol. 2, Springer, Berlin, 1890, pp. 133–143. First published in Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, volume 15, 1870, pp.272–286.

[22] C. L. Seitz, J. Seizovic, and W.-K. Su, *The C programmer's abbreviated guide to multicomputer programming*, Tech. Report Caltech-CS-TR-88-1, Dept. of Computer Science, California Institute of Technology, January 1988.

[23] O. B. Widlund, *On the rate of convergence of the classical Schwarz alternating method in the case of more than two subregions*, tech. report, Department of Computer Science, Courant Institute, 1989. A revised version in preparation.

[24] ——, *Optimal iterative refinement methods*, in Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1989, SIAM.

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