CHAPTER 30

Domain Decomposition Algorithms of Schwarz Type, Designed for Massively Parallel Computers*

P. E. Bjørstad†
M. D. Skogen†

Abstract. We discuss implementation of additive Schwarz type algorithms on SIMD computers. A recursive, additive algorithm is compared with a two-level scheme. These methods are based on a subdivision of the domain into thousands of micro-patches that can reflect local properties, coupled with a coarser, global discretization where the 'macro' behavior is reflected. The two-level method shows very promising flexibility, convergence and performance properties when implemented on a massively parallel SIMD computer.

Key words. Domain decomposition, Schwarz algorithm, Massively parallel algorithms, Elliptic partial differential equations, SIMD computers.

1. Introduction. The use of domain decomposition algorithms have been subject to active research in the last few years [10, 11, 12, 17]. A strong motivation for this has been the increasing importance of parallel computers in scientific computing. The theoretical foundation of these methods has been extended from the two domain case [1, 4] to situations where the number of subdomains can be very large [8, 9, 15]. Until now, most computational experience has been obtained from using a relatively small number of subdomains implemented on both shared and local memory MIMD computers [2, 20]. The need for global coupling and exchange of information between individual subdomains in every iteration, is also widely appreciated [8, 9, 15]. As the number of subdomains increases, this issue raises a number of questions when an efficient parallel implementation is desired [2, 20]. In this paper, we propose and demonstrate a highly efficient algorithm designed for SIMD computers having thousands of processors. The number of subdomains can easily be of the same order as the number of processors without incurring excessive cost from the global solver.

*This work was supported in part, by NAVF, grant 413.89/024 and in part by NTNF, contract IT0228.20484.
†Institutt for Informatikk, University of Bergen, Høyteknologisenteret, N-5020 Bergen, Norway.

We first very briefly describe the fundamental ideas of domain decomposition algorithms based on a Schwarz type iteration [23], in particular the additive version [15]. We proceed to discuss and compare implementations of such algorithms on SIMD computers. We conclude that a two-level algorithm combining inexact solvers on the subdomains with an inexact multilevel iteration on the global problem, has favorable characteristics for massively parallel computer systems. This class of algorithms should work well on a wide range of parallel machines. There is considerable freedom to choose specific components in the algorithms, and to optimize performance on a given computer architecture. In particular, these methods can be tailored to both MIMD and SIMD machines. We focus on SIMD computers in this paper, reporting on experiments carried out on a MasPar MP-1. We therefore give a brief description of this machine in Section 3.

2. Global Additive Schwarz, GAS. We will in this section, give a brief description of the Global Additive Schwarz (GAS) algorithm, since this method will serve as the basis for the algorithms to be discussed in Sections 4 and 5. For a more detailed exposition and analysis of this method see [14, 15].

Let V be a Hilbert space, $a(\cdot,\cdot)$ a symmetric, strongly elliptic bilinear form and f a linear form. Our starting point is the variational problem:

$$\text{Find } u \in V \text{ such that}: \ a(u,v) = f(v) \ \forall v \in V, \tag{1}$$

and its discrete approximation:

$$\text{Find } u_h \in V_h \text{ such that}: \ a(u_h,v_h) = f(v_h) \ \forall v_h \in V_h, \ V_h \subset V. \tag{2}$$

We look for solutions of the form

$$u_h = \sum_{i=0}^{N} u_i, \tag{3}$$

where

$$a(u_i,v_i) = f(v_i) \ \forall v_i \in V_i, \ V_i = V_h \cap V(\Omega_i), \ \ i \geq 1. \tag{4}$$

Here Ω_i describes the 'micro'-patches (or substructures) of our domain Ω. This subdivision can also be viewed as a coarse discretization of Ω (with mesh size H equal to the diameter of Ω_i). We define the coarse space problem needed in (3), by

$$a(u_0,v_0) = f(v_0) \ \forall v_0 \in V_0, \ V_0 = V_H \cap V(\Omega). \tag{5}$$

We assume that the subdomains Ω_i have a (possibly small) overlap with their nearest neighbors.

Defining the projections $P_i : V_h \to V_i$, we observe that u_h can be found by solving

$$Pu_h \equiv (\sum_{i=0}^{N} P_i)u_h = g_a, \tag{6}$$

where g_a can be computed since

$$a(P_i u_h, v_h) = f(v_h) \ \forall v_h \in V_i, \tag{7}$$

and each term in the sum (6) can be computed independently. We note that P is symmetric in the inner product defined by the bilinear form $a(\cdot,\cdot)$ and that forming a product of the form $w_h = Pv_h$, requires the solution of (4) and (5).

Turning now to a description of the algorithm in matrix terms only, we characterize the projections introduced above. If we denote the contribution from subdomain Ω_i (by subassembly) to the global stiffness matrix K by $K^{(i)}$ and permute the unknowns appropriately, then

$$y = P_i x = \begin{pmatrix} K^{(i)^{-1}} & 0 \\ 0 & 0 \end{pmatrix} Kx. \tag{8}$$

¿From (8) one can also interpret the method as a preconditioner for the original stiffness matrix K. Note in particular, that the case $i = 0$ corresponds precisely to a second level substructure in the language of structural engineering, but unlike the standard approach in today's structures codes, we include these degrees of freedom also in the substructure problems.

It is well known [15] that the formulation based on (6) can lead to an iteration operator with a uniformly bounded condition number. A simple first order Richardson iteration based on this splitting, takes the form

$$u_h^{n+1} = u_h^n - \tau(Pu_h^n - g_a) \tag{9}$$

where τ is a positive constant that depends on the eigenvalues of the iteration operator, the optimal value being $\tau = 2/(\lambda_{min} + \lambda_{max})$. In practical algorithms we often accelerate this scheme with the conjugate gradient method. The method converges to within a prescribed tolerance, independent of the discretization parameter h and of the number of substructures N. We note that each iteration requires the solution of N local problems on the substructures and the solution of the coarse problem, all of which are independent and can be solved in parallel. Observe that we can replace both the subdomain solution (4) and the coarse problem solution (5) with inexact solutions without destroying the optimal rate of convergence, provided that the new problems (replacing (4) and (5)) are spectrally equivalent to the bilinear form $a(u,v)$ restricted to the appropriate space $V_i \times V_i$.

Our concern in this paper, is to discuss efficient implementations of this algorithm on massively parallel systems. The interaction between the local problems and the coarse (global) problem is in this context, the critical issue.

3. A brief description of the MasPar MP-1 Computer. The MasPar MP-1 system is a massively parallel SIMD computer system. The system consists of a high performance UNIX workstation (FE) and a data parallel unit (DPU). The DPU consist of at least 1024 processor elements (PEs) each with local memory and register space. All processors execute instructions broadcast by an array control unit (ACU) in lockstep, but each processor can disable itself based on logical expressions for conditional execution. It should be noted that the individual processors may operate not only on different data, but also in different memory locations, thus supporting an indirect addressing mode.

There are three communication mechanisms available, the Xnet, the router and the global or-tree.

The Xnet interconnects the PEs in a 2 dimensional toroidal mesh which also allows for diagonal communication. The Xnet operates in three modes, Xnet, XnetP(ipe) and XnetC(opy). Xnet has a cost proportional to the size of the message times the distance of transmission. XnetP and XnetC are pipelined versions with a cost that only depends on the sum of the message length and the distance. The two last modes are useful for regular, non-local communication, but require that the processors between any pair of communicating processors be inactive. Thus, for sending over longer distance XnetP is much faster than basic Xnet provided that it can be used. XnetC is similar to XnetP, leaving a copy of the transmitted variable on all the intermediate processors. The MP-1 also supports arbitrary node to node communication through a three-stage switch called the router. The global or-tree can move data from the individual processors to the ACU. If many processors send data at the same time a global reduction results.

Floating point is implemented in software. We define the average time of a floating point instruction, $\alpha = \frac{1}{2}(Mult + Add)$. Measured in units of α, the floating point performance of the MP-1, corresponds to a peak speed of 290 Mflops on a machine having 8192 processors. The time ratio of nearest neighbor communication to floating point computation defined as $\gamma = Xnet[1]/\alpha$, has $\gamma \approx 1/5$ for 64 bit operands.

A more detailed general description of the MasPar MP-1 computer can be found in [6, 13, 22].

4. Recursive Global Additive Schwarz, RGAS.

4.1. Description of the method. The idea with RGAS is to solve the coarse problem recursively using GAS. A convergence analysis of RGAS is given in [16]. In order to define a recursion, we divide the coarse problem into subdomains in the standard fashion. Applying additive Schwarz to this problem, we can now define a new (and smaller) coarse problem having the intersection of the boundaries of the new subdomains as unknowns. Since the coarse problem in GAS can be solved in parallel with all the interior subdomain solves, this strategy can potentially increase the parallel part of the algorithm.

The approach is very similar to multigrid, and in order to appreciate this we compare a two-level multigrid iteration (10) with the GAS algorithm (11).

(10)
$$\begin{tikzcd}
u_h^j \arrow[r, "relax"] & \bar{u}_h \arrow[r] & r_h = f_h - K_h\bar{u}_h \arrow[d, "I_h^H"] & e_h + \bar{u}_h = \bar{\bar{u}}_h \arrow[r, "relax"] & u_h^{j+1} \\
 & & r_H \arrow[r] & K_H e_H = r_H \arrow[r] & e_H \arrow[u, "I_H^h"]
\end{tikzcd}$$

$$u_h^j \to r_h = f_h - K_h u_h^j \to (K_h^{ii} e_h^i = r_h^i)_{i=1}^N \to u_h^{j+1} = u_h^j + \tau \sum_{i=0}^N e_h^i$$

(11)
$$\begin{tikzcd} {} \arrow[d, "I_h^H"] & & {} \\ r_H \arrow[r] & K_H e_H^0 = r_H \arrow[r] & e_H^0 \arrow[u, "I_H^h"'] \end{tikzcd}$$

Here K_h is the stiffness matrix, K_H refers to the stiffness matrix for the coarse problem, while f and τ are defined in (7) and (9). The restriction and interpolation between the two levels are of the same form in the two algorithms. The multigrid relaxation has been replaced with subdomain solves. Knowing the purpose of multigrid relaxation, it is no surprise that the subdomain solves in (11) should be replaced by approximate solvers in order to minimize the computational work. A natural choice is just to do a few Gauss-Seidel sweeps, making the actual computational procedure look even more like multigrid. The possible advantage of the GAS algorithm with respect to parallel execution, is that one in (11) can solve the problems on the two levels in parallel, while the multigrid algorithm (10) is strictly sequential between levels.

The generalization of GAS to get RGAS is straightforward from (11), just as one can derive various multigrid iterations from (10) by recursively using the same approach to solve the coarse problem defined by K_H. The structure of an $L+1$ level 'V-cycle' version of the RGAS algorithm is shown in (12).

(12)
$$\begin{tikzcd} u_0^j \arrow[r] & r_0 = f_0 - K_0 u_0^j \arrow[r] \arrow[d, "I_0^1"] & (K_0^{ii} e_0^i = r_0^i)_{i=1}^{N_0} \arrow[r] & u_0^{j+1} = u_0^j + \tau \sum_{i=0}^{L} I_i^0 e_i \\ & r_1 \arrow[r] \arrow[d, phantom, "\vdots"] & (K_1^{ii} e_1^i = r_1^i)_{i=1}^{N_1} \arrow[r] & e_1 \arrow[u, "I_1^0"'] \\ & {} \arrow[d, "I_{L-1}^L"] & & {} \arrow[u, phantom, "\vdots"] \\ & r_L \arrow[r] & K_L e_L = r_L \arrow[r] & e_L \arrow[u, "I_L^{L-1}"'] \end{tikzcd}$$

As in multigrid methods there are many possible variants. (Different cycles, different approximate solvers etc.) In the resulting RGAS method, one can solve all the problems $K_k^{ii} e_k^i = r_k^i \;\; i = 1, 2, \ldots N_k$ at all levels $k = 0, 1, \ldots L$, simultaneously. The size of the subproblems may change from one level to the next. Note that in (12) the interpolation and restriction may be implemented 'sequentially' between successive levels or 'directly' between any level and the finest level. Other combinations are also possible and may be of interest depending on the target computer architecture. Unfortunately, on an SIMD computer, the setup of the problems on all levels and the interpolation and restriction must typically be implemented as sequential steps also in the RGAS algorithm.

Assuming that the number of grid points can be chosen carefully, one can also arrange the size of the subdomains to be almost the same on all levels in the recursion. Therefore, in each step, we can solve many small (identical) subproblems in parallel, which in our SIMD context, may be more attractive than first solving many small subproblems and then one global coarse grid problem. Finally, we note (see [16]) that the BPX algorithm [7] may be viewed as a limiting case of (12) where the subdomains have degenerated to the support of the individual basis functions and that similarly, certain multigrid algorithms can be interpreted as multiplicative variants, updating the residual at each level.

4.2. Implementation. An important motivation for RGAS is to avoid having to solve a coarse problem of possibly much larger size than the subdomain problems. In an SIMD implementation the main concern will always be to keep processors busy and to minimize communication cost. The first consideration requires ideally that all subdomains be of precisely the same size and structure. This is not possible since the subdomains near the boundary will have a different size or different overlap from the interior ones. Due to the lack of overlap at the exterior boundary, one can either increase the overlap from these substructures to their interior neighbors and thus keep the subdomain size fixed, or let these subdomains be slightly smaller in size. This problem can be handled somewhat more easily with the use of an iterative solver instead of a direct subdomain solver. An iterative solver also allows for inexact subdomain solutions which as we shall see, are preferable to the exact solver. In this report, we use a symmetric Gauss-Seidel iteration. The overlap will in all our experiments, be just one common line of grid points, that is, as the mesh size is refined we use a relatively smaller overlap. This choice is based on previous experience [2] that clearly shows this to be optimal with respect to computer time. The choice also minimizes the variation in size of the subdomains. It should be remarked that the use of a coarse grid problem combined with relatively many subdomains are necessary for these conclusions to hold. The Schwarz algorithm without the coarse problem is much more sensitive to the size of the overlap [2]. Note also that even in our case, the number of iterations required can decrease slightly with an increase in the overlap, but this is offset by a higher computational cost per iteration.

The main advantage of RGAS is the ability to solve all the subproblems on all levels simultaneously in an efficient SIMD fashion. Unfortunately, the method leaves very little flexibility with respect to the size of the substructures. Also, the problem of assigning substructures from the coarser level problems to processors in a way that keeps both the programming simple and the communication cost low, is not easy. On our machine with 2^{13} processors, we have assigned half the processors to the original substructures, using the other half to solve the subproblems arising from the coarse problem. Thus, our processor utilization is always less than 0.75. If we have more substructures than processors, this can be improved, but not without a considerably more sophisticated implementation that easily could lead to loss of efficiency.

4.3. Numerical experiments with RGAS. We consider a test problem obtained from (1) with

$$a(u,v) = \int_\Omega k(x,y)\nabla u \nabla v \, d\Omega\,, \tag{13}$$

corresponding to the classical equation

$$\begin{aligned} \nabla \cdot k(x,y)\nabla u = {} & f(x,y) && \textit{in } \Omega \\ u = {} & g(x,y) && \textit{on } \partial\Omega \end{aligned} \tag{14}$$

defined on the unit square $\Omega = (0 \leq x, y \leq 1)$. We assume that $k(x,y)$ is positive and piecewise continuous, but we are interested in the case where this function has jump discontinuities. We take $V = H_0^1(\Omega)$ and define the discrete space V_h using uniform triangles and P_1 finite elements (corresponding to the 5-point stencil). The functions $f(x,y)$ and $g(x,y)$ have been picked such that the exact solution of (14) is $u = x^2 + y^2 - x \exp x \cos y$ when $k(x,y) \equiv 1$. This problem was also considered in [4, 5]. In all experiments reported in this paper, we stop the iteration when the discrete L_2 norm of the residual has been reduced by a factor 10^{-6}.

In Table 1 we compare GAS and RGAS using an exact solver in RGAS, in order to have a reference to the cases to be reported next, where we use a symmetric Gauss-Seidel iteration. We used 8192 processors for the RGAS entry, storing all the fine

Mesh size	$h = 1/128$	$h = 1/256$	$h = 1/512$
Algorithm	Iterations	Iterations	Iterations
RGAS	26	20	20
GAS	13	13	14

TABLE 1
Iterations for RGAS and GAS with exact subdomain solvers.

Subdomain size	3 x 3	5 x 5	9 x 9
Restriction	0.13	0.09	0.12
Solve	0.01	0.04	0.20
Interpolation	0.15	0.12	0.15

TABLE 2
Comparison of the times for solve, interpolation and restriction in RGAS.

Subdomain size	3 x 3		5 x 5		9 x 9	
Mesh size	Cond.	# Iter.	Cond.	# Iter.	Cond.	# Iter.
$h = 1/32$	12.4	22	8.6	18*	7.6	16*
$h = 1/64$	13.6	24	8.7	19	7.6	16
$h = 1/128$	14.7	26	9.9	20*	9.6	19*
$h = 1/256$	15.4	27	9.9	20	9.6	20*
$h = 1/512$	16.1	27	10.7	21*	9.7	20

TABLE 3
Convergence properties of RGAS

level subdomains in half of the machine. We observe that RGAS requires up to twice as many iterations as GAS due to the inexact solution of the coarse problem. ¿From previous experience with GAS on MIMD machines [2], we note that the size of the coarse problem relative to the size of the subdomain problems is important. The solution of the coarse and the fine problems can be performed in parallel and usually accounts for more than ninety percent of the total computing time. With RGAS, we are able to solve all the problems in SIMD parallel fashion thus reducing this relative time. It turns out that we need to do about three symmetric Gauss-Seidel iterations on the subproblems in each iteration, to minimize computer time.

The disadvantage of RGAS is, besides the inflexibility described earlier, that new 'sequential' steps have been introduced in the algorithm. The interpolation and restriction operations in (12) are now more expensive than in GAS, and many processors will be idle during this step, contributing to a lower overall processor utilization. The work is performed in a sequential manner with respect to the levels and also involves irregular and non-local interprocessor communication. The interpolation and restriction work is insignificant in GAS, but as seen in Table 2, it is significant relative to the work of solving the subproblems in RGAS. The entries in Table 2 are taken from the example studied further in Table 7. The above shortcomings of RGAS outweighs the attractive features and lead us to consider an alternative class of algorithms in the next section.

In Table 3, we study the convergence behavior of RGAS as a function of both subdomain size and the number of levels in the recursion. Note that we make a distinction between our domain being divided into substructures and the size of the subdomains which includes the unknowns in the overlap region. Thus, the size (unknowns) of a subdomain times the number of substructures is larger than the number of unknowns in the original problem. If the recursion is terminated early (by solving a 'larger' coarse problem at the lowest level) this improves the condition number of the iteration operator and therefore reduces the number of iterations. In the SIMD context, it is of interest to carry the recursion all the way until we have a single problem of smaller or equal size at the final level of the algorithm. We see from the table that the number of iterations required to solve our test problem, appears to be independent of the number of levels. Despite this, we observe a slow increase in the estimated condition number of the iteration operator. This estimate was computed based on the conjugate gradient process in the usual way [18], but it was observed that up to twice as many iterations (compared to what is given in the table for the required residual reduction) were needed in order to get a reliable estimate for the smallest eigenvalue. The computation confirms that the smallest eigenvalue is bounded from below [16] and possibly even increases slightly. The largest eigenvalue accounts for the slight increase in the condition number. We note that the finest mesh size and the smallest subdomain in Table 3 requires 8 levels of recursion. The entries marked with a star correspond to cases where the 'last' problem had a smaller size, this may account for the slightly irregular behavior. The computation indicates that the number of iterations needed is independent of the number of levels. With this assumption the RGAS algorithms have optimality properties similar to those of multigrid methods.

```
repeat
    all update local residual-vector
    all compute local $\alpha_k$
    GLOBADD(local $\alpha_k$) to global $\alpha_k$
    compute $\beta_k$
    all update local search-vector ($p$)
    all compute local $Kp$ (rhs) for the preconditioner
        (GET 'boundary' from neighbors,
        compute local $Kp$)
    start preconditioning
        all generate local node for coarse problem
            (find restriction,
            add up to local coarse node)
        all perform Sym-GS (3) iterations on local problem
        perform MG-cycles (3) on coarse problem
        all add solutions from local domains
            (GET overlap from neighbors,
            add solutions)
        all interpolate coarse grid to fine grid
    end preconditioning
    find global $a_k$ (step length, as $\alpha_k$)
    all update local solution-vector
until convergence
```

FIG. 1. *CG-accelerated MGAS, one domain per processor*

In fact, it was announced at this meeting [24] that Mr. Xuejun Zhang has proved an upper bound independent of the number of levels, thus improving on the estimates given in [16].

5. Multilevel Global Additive Schwarz, MGAS.

5.1. Description of the method.

Based on the experience in the previous section, we abandon the idea of solving all RGAS subproblems in one parallel step. A more flexible algorithm can be developed based on the original GAS algorithm in (11). We first (approximately) solve all subproblems on the finest mesh in one parallel step. We then consider the solution of the coarse problem separately. This can again be solved with RGAS or with a standard multigrid algorithm. One could also consider hybrid schemes, possibly truncated before reaching the final level of recursion. In this way, we are able to use all processors better on the fine grid and also create a more efficient data structure for the coarse problem as well as for the restriction and interpolation. By using a 'sequential' two-level scheme, we may also consider to link the two levels using 'multiplicative Schwarz' [2] instead of the additive method. We know that this method in general, has better convergence properties than the additive algorithm. Keyes and Gropp [21] advocate a two-level algorithm in quite a different context, but some of the underlying motivation related to simple, flexible data structures and efficient parallel implementations, is similar.

To illustrate the method, we here report on an additive implementation where the coarse problem is solved approximately using three multigrid V-cycles. A single conjugate gradient iteration of MGAS is outlined in Figure 5.1. As seen from the algorithm the parallelism is very high. We have full processor utilization in all steps except in the computation of inner products and in the multigrid cycle.

Subdomain size	# Sym. GS Iter.	Time	Iterations	Exact local	Exact global
$3x3$	1	2.2	15	14	15
$5x5$	3	3.5	15	14	15
$7x7$	3	6.0	17	14	17
$9x9$	3	9.9	19	15	19
$11x11$	3	16.0	22	17	22
$13x13$	3	23.0	25	17	25
$15x15$	3	34.0	28	18	28
$17x17$	3	47.0	31	19	32

TABLE 4
Number of iterations, inexact vs. exact solvers

5.2. Implementation. We have implemented an MGAS code on our MasPar MP-1 using 4096 or 8192 processors [1]. We typically store 1-2 local domains in each processor and 1-2 points from the coarse problem. The code solves (14), possibly with discontinuous $k(x,y)$. We remark that a code of this kind can be implemented more efficiently if one restricts the function $k(x,y)$ to be a constant on each subdomain. This may be permissible in many applications as we can use thousands of subdomains. In the results reported here, we take advantage of this option.

We have chosen to use a symmetric Gauss-Seidel iteration as our approximate solver on the subdomains, and a fixed number of standard multigrid V-cycles for the solution of the coarse grid problem. We have also fixed the number of relaxation sweeps in the multigrid cycle to $\nu_1 = \nu_2 = 2$. In Table 4 we have given the number of Gauss-Seidel iterations that minimizes the total elapsed time, and compared this with the number of iterations needed for convergence using an exact solver for the local subproblems. In the last column we list the number of iterations needed when we keep the inexact solution of the subproblems, but solve the coarse problem with an exact solver. The test problem is (14) with $k(x,y) \equiv 1$ and the same stopping criterion as before. We have used 4096 processors with one subdomain per processor. We see that the optimal number of Gauss-Seidel iterations is close to three, similar to the observation when using RGAS. We further observe the effect of using a relatively smaller overlap when increasing the subdomain size, in the column 'Exact local'. The larger increase in the iteration count when using the symmetric Gauss-Seidel iteration, reflects the poorer quality of the inexact solver as the subdomain grid is refined. Clearly, for very large subdomains, one should replace this iteration with a locally optimal method like a multigrid V-cycle, in order to maintain a global work complexity proportional to the number of unknowns.

5.3. Complexity. We can estimate the total work in this MGAS algorithm as a function of some of the parameters. Let n_1 be the number of Gauss-Seidel iterations, n_2 the number of V-cycles, each doing $\nu = \nu_1 + \nu_2$ relaxation sweeps. If as before, we have N subdomains on the finest grid each having m^2 unknowns then the leading

[1] A prize winning algorithm in the Mannheim SuParCup '91 competition [3]

	m=3	m=5	m=9	m=17
Global solver	65	42	19	6
Local solver	18	28	36	40
Conjugate gradient	10	15	20	23
$y = Kx$	3	5	6	6
Interpolation	3	6	10	13
Restriction	1	4	9	12

TABLE 5
Estimated relative time for different parts of an MGAS algorithm.

	m=3	m=5	m=9	m = 17
ρ	0.09	0.06	0.03	0.01

TABLE 6
Estimated ratio communication to computation

term of the arithmetic work per iteration, takes the form

$$c_1 m^2(n_1 + c_2) + c_3 n_2(\nu + c_4)\log_2 N, \tag{15}$$

while the communication behaves like

$$c_5 m + c_6 n_2(\nu + c_7)\log_2 N. \tag{16}$$

Observe, that unlike the RGAS algorithm in the previous section, we are now able to take much better advantage of the fast, pipelined communication provided by XnetP. This communication mechanism is very important on the coarser levels of a standard multigrid or RGAS algorithm employed in our two level MGAS scheme. Using the actual values of the constants in (15) and (16), and the values $n_1 = n_2 = 3$, $\nu = 4$, and $N = 4096$, we have estimated the relative cost of the different parts of this MGAS algorithm in Table 5. Similarly, in Table 6, we list the ratio ρ of communication to arithmetic. Actual time measurements agree quite well with this model. Note in particular, the very low communication cost that this algorithm combined with the MP-1 achieves.

5.4. Numerical experiments with MGAS. In Table 7, we compare MGAS using 4096 processors each having one subdomain, with RGAS running on 8192 processors, but storing the fine grid subproblems on half the processors. The running time has been minimized by finding the best number of Gauss-Seidel iterations for each of the six cases. The size of the subproblems is $m = 3$, 5 and $m = 7$, all with $k(x, y) \equiv 1$. We notice that MGAS is superior despite the use of fewer processors.

In the next table, Table 8, we investigate the robustness of MGAS when solving (14) with discontinuous $k(x, y)$. We solve a problem having 16384 subdomains with four different sizes for the local subproblem. In this table, we fix the number of inner Gauss-Seidel iterations at three, a number which is close to optimal over a range of problems we have considered. The first problem has $k \equiv 1$ for reference, the

Mesh	$h = 1/128$		$h = 1/256$		$h = 1/512$	
Method	# Iter.	Time	# Iter.	Time	# Iter.	Time
RGAS	26	5.9	23	6.5	25	15.6
MGAS	15	2.2	15	3.5	19	9.9

TABLE 7
Comparing RGAS and MGAS

Mesh size	$h = 1/256$		$h = 1/512$		$h = 1/1024$		$h = 1/1408$	
$k(x,y)$	# Iter.	Time	# Iter.	Time	# Iter.	Time	#Iter.	Time
1	15	4.2	14	6.1	19	18.4	23	35.5
$10(x^2+y^2)+.01$	15	4.2	15	6.6	19	18.4	24	37.0
'Greenbaum'	15	4.2	15	6.6	19	18.4	25	38.6
Checkerboard	15	4.2	16	7.0	24	23.3	30	46.5
Random	15	4.2	16	7.0	22	21.3	27	42.0

TABLE 8
MGAS for different $k(x,y)$ using 8192 processors and 16384 subdomains.

second entry shows a variable $k(x,y)$ where the computation proceeds with a 'frozen' constant value of k within each subdomain. The error compared to the solution of the continuous problem was less than $3 \cdot 10^{-4}$ on all grids. The third entry is taken from [19] where the global domain is cut into four smaller squares. The SW and NE square has $k = 1$, the NW has $k = 10^4$, and the SE square has $k = 10^{-4}$. Our next entry is a checkerboard (with 16384 squares!) where all the 'black' domains have $k = 1$, while the 'white' domains have $k = 10^3$. In the last entry of Table 8 each of the 16384 subdomains has been assigned a constant, but random value between 1 and 1024, for $k(x,y)$. The results show that the algorithm is very robust and that we are able to solve a problem with strong discontinuities, having two million unknowns, in approximately 40 seconds of computing time.

Finally, in Table 9, we compare the performance of MGAS on the MP-1 with the equivalent algorithm implemented on vector machines. That is, an implementation that vectorizes across all the subproblems. We used an Alliant FX/8 with 8 vector processors and a CRAY X-MP/2, but only using one processor. We have solved the problem with 1024, 4096 and 16384 subdomains, each with $m = 5$ (5×5 points in each subdomain). On the MP-1, we use one processor per domain except for the largest case where we map two subdomains to each processor. Again, we take the number of Gauss-Seidel iterations for each entry in the table that minimizes the computing time. All machines run the same algorithm, but the differences in the iteration counts show that the Alliant prefers fewer inner iterations. The table clearly shows how the time scales with the problem size for the three machines. The increase for the MP-1 when solving the largest problem, is due to the extra work of handling two subproblems on each processor.

# Subdomains	1024		4096		16384	
Computer	# Iter.	Time	# Iter.	Time	# Iter.	Time
MasPar MP-1	15	3.5	15	3.6	14	6.0
Alliant FX/8	16	4.9	18	19.9	18	84.3
CRAY X-MP	14	1.3	14	4.8	14	18.2

TABLE 9
SIMD machine compared to vector machines

6. Conclusions. We conclude that algorithms of the MGAS family, can be efficiently implemented on massively parallel SIMD computers. Our experiments show that one can construct efficient domain decomposition algorithms using thousands of subdomains, and that combined with SIMD computers, this results in a very cost effective way of solution. The method is quite insensitive to a discontinuous $k(x,y)$, and therefore promising within application areas like oil reservoir simulation.

This class of algorithms should be further studied with respect to three dimensional problems, more general geometries, the case of local grid refinement, and with respect to nonsymmetric operators. The possibility of using the indirect addressing supported on machines like the MP-1, in order to increase the generality and flexibility of these algorithms on massively parallel computers should also be considered.

7. Acknowledgement. The first author did part of this work while visiting The Courant Institute of Mathematical Sciences. He thanks Maksymilian Dryja and Olof Widlund for interesting discussions.

REFERENCES

[1] P. E. BJØRSTAD AND J. MANDEL, *On the spectra of sums of orthogonal projections with applications to parallel computing*, BIT, 31 (1991), pp. 76–88.

[2] P. E. BJØRSTAD, R. MOE, AND M. SKOGEN, *Parallel domain decomposition and iterative refinement algorithms*, in Parallel Algorithms for Partial Differential Equations, W. Hackbusch, ed., Braunschweig, Germany, 1991, VIEWEG. Notes on Numerical Fluid Mechanics, Vol 31, Proceedings of the sixth GAMM-Seminar, Kiel, January 19-21 1990.

[3] P. E. BJØRSTAD AND M. D. SKOGEN, *A new, massively parallel algorithm for the solution of elliptic equations with discontinuous coefficients*, Tech. Report, Institute of Informatics, University of Bergen, Thormøhlensgate 55, N - 5008, Bergen, Norway, 1991. Submitted to the Mannheim '91 SuParCup competition.

[4] P. E. BJØRSTAD AND O. B. WIDLUND, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1093–1120.

[5] ——, *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.

[6] T. BLANK, *The MasPar MP-1 architecture*, in Proceedings of IEEE Compcon Spring 1990, IEEE, February 1990.

[7] J. BRAMBLE, J. PASCIAK, AND J. XU, *Parallel multilevel preconditioners*, in Domain Decomposition Methods for Partial Differential Equations III, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1990, SIAM. Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, Houston, Texas, March, 1989.

[8] J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring, III*, Math. Comp., 51 (1988), pp. 415 – 430.

[9] ——, *The construction of preconditioners for elliptic problems by substructuring, IV*, Math. Comp., 53 (1989), pp. 1 – 24.

[10] 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.

[11] ——, eds., *Domain Decomposition Methods for Partial Differential Equations III*, Philadelphia, 1990, SIAM. Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, Houston, Texas, March, 1989.

[12] ——, eds., *Domain Decomposition Methods for Partial Differential Equations IV*, Philadelphia, 1991, SIAM. Proceedings of the Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations, Moscow, May, 1990.

[13] P. CHRISTY, *Software to support massively parallel computing on the MasPar MP-1*, in Proceedings of IEEE Compcon Spring 1990, IEEE, February 1990.

[14] M. DRYJA, *An additive Schwarz algorithm for two- and three- dimensional finite element elliptic problems*, in Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1989, SIAM.

[15] M. DRYJA AND O. B. WIDLUND, *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.

[16] ——, *Multilevel additive methods for elliptic finite element problems*, in Parallel Algorithms for Partial Differential Equations, W. Hackbusch, ed., Braunschweig, Germany, 1991, VIEWEG. Notes on Numerical Fluid Mechanics, Vol 31, Proceedings of the sixth GAMM-Seminar, Kiel, January 19-21 1990.

[17] 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.

[18] G. H. GOLUB AND C. F.VAN LOAN, *Matrix Computations*, Johns Hopkins Univ. Press, 1989. Second Edition.

[19] A. GREENBAUM, C. LI, AND H. Z. CHAO, *Parallelizing preconditioned conjugate gradient algorithms*, Computer Physics Communications, 53 (1989), pp. 295–309.

[20] D. E. KEYES AND W. D. GROPP, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. s166 – s202.

[21] ——, *Domain decomposition with local mesh refinement*, Tech. Report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, 1990. To appear in SIAM Journal on Scientific and Statistical Computing.

[22] J. NICKOLLS, *The design of the MasPar MP-1, a cost effective massively parallel computer*, in Proceedings of IEEE Compcon Spring 1990, IEEE, February 1990.

[23] 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.

[24] X. ZHANG, *A multilevel additive Schwarz method*, Tech. Report, Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012, 1991. Announced at the Fifth International Symposium on Domain Decomposition Methods, Norfolk, Virginia 1991.