1992

M. Louter-Nool

MGD1M, a parallel multigrid code with a fast vectorized ILU-relaxation

Department of Numerical Mathematics Report NM-R9222 November

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

MGD1M, a Parallel Multigrid Code with a Fast Vectorized ILU-Relaxation

Margreet Louter-Nool <greta@cwi.nl>
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
phone: +31 20 5924101, fax: +31 20 5924199

Abstract. In this paper we focus on MGD1M, a multigrid method which uses Incomplete LU-relaxation (ILU) as a smoothing process. A straightforward ILU implementation leads to non-vectorizable recursive relations. By applying an alternative ordering of the grid points, proposed by Ashcraft et al.[1], the ILU vectorizes and the overall execution time is reduced considerably.
We also consider general techniques to improve the (vector) performance on the Cray Y-MP4, which are also suitable for other high-performance machines. Parallelism has been introduced by applying auto- and macrotasking. Performance results measured on a dedicated Cray Y-MP4/464 are discussed.

1991 Mathematics Subject Classification: 65N55, 65F10, 65Y05.
1991 CR Categories: G.1.8, G.1.3, G.1.0.

Keywords & Phrases: Elliptic PDEs, ILU-relaxation, multigrid methods, numerical software, parallel computers, sparse linear systems, vector computers.

Note: The implementation is available in ANSI Fortran 77. A variant using macro- and autotasking tuned for the Cray Y-MP is available, too.

1. Introduction

In this paper we discuss a new implementation of MGD1V (see [7,10,11]), a multigrid method for the solution of linear systems resulting from the 7-point discretization of a general linear second-order elliptic partial differential equation in two dimensions. Although MGD1V is not the best choice to obtain high performances on a parallel multiprocessor machine like the Cray Y-MP, its use can be recommended, since it is a very fast converging method.

When we started with this project, a vectorized version was already available (see [7,12]). This is a portable implementation operating on very long one-dimensional vectors which is well suited for memory-to-memory machines like the Cyber 205. As a matter of fact, this vectorized version runs also pretty well on one processor of the Cray Y-MP. To make a clear difference between the vector version MGD1V by De Zeeuw[12] and the new Multitasked variant, we will refer to the latter as MGD1M. The algorithm is briefly described in Section 2. Section 3 concerns the Cray-tools we used to analyze the performance. In Section 4, we make some alterations to improve the vector performance. To obtain the highest possible efficiency on the Y-MP we apply auto- and macrotasking[2,4]. The results are discussed in Section 5. Finally, in Section 6 we comment on the total speedup obtained by vectorization and parallelization.

Report NM-R9222
ISSN 0169-0388
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

2. THE ALGORITHM

The multigrid method used by MGD1V and MGD1M has been extensively discussed in [6,7,11,13]. Here we restrict ourselves to a short review. We assume a discretization using a regular Courant triangulation of a rectangle. In this way 7-point discretization stencils are obtained. An advantage of this stencil over 5-point stencils is that mixed derivatives can be represented. The algorithm uses discretizations on a sequence of increasingly coarser grids Ω^k, $k = l, l-1, \ldots, 1$. The user has to define the discretization of the matrix A_l and the right hand side f_l only on the finest grid Ω^l. The particular multigrid algorithm for the solution of

$$A_l\, u_l = f_l \tag{2.1}$$

consists of an adequate number of so called sawtooth multigrid cycles. The discrete operators A_k corresponding with the coarser grids Ω^k are automatically derived by the program. MGD1M uses Incomplete LU-decomposition as relaxation procedure because of its excellent smoothing properties. The code MGD1M as a whole performs satisfactory for equations like the convection-diffusion equation and the anisotropic diffusion equation. On each level the 7-diagonal matrix A_k is decomposed as

$$A_k = L_k\, U_k - C_k \tag{2.2}$$

where L_k is a lower-triangular matrix (with unity on the main diagonal) and U_k is an upper-triangular matrix. L_k and U_k are allowed to have non-zero diagonals only where A_k has. The remainder matrix C_k has only two non-zero diagonals. For more details on the incomplete decomposition we refer to Section 4.2. One relaxation sweep of the ILU-relaxation corresponds to the solution of the system

$$L_k\, U_k\, u_k^{(i+1)} = f_k + C_k\, u_k^{(i)}. \tag{2.3}$$

Consequently, the residual is computed efficiently by

$$r_k^{(i+1)} = f_k - A_k\, u_k^{(i+1)} = C_k\, (\, u_k^{(i+1)} - u_k^{(i)}\,). \tag{2.4}$$

In the following we outline the global structure of MGD1M. In a preparational phase, a sequence of coarse grid operators is constructed in subroutine RAP by means of subsequent Galerkin approximations:

$$A_{k-1} = R_{k-1,k}\, A_k\, P_{k,k-1}, \qquad k = l, l-1, \ldots, 2, \tag{2.5}$$

where the restriction operator $R_{k-1,k}$ transfers a grid function onto the next coarser level $k-1$ and the prolongation operator $P_{k,k-1}$ denotes the interpolation from level $k-1$ to level k (see Figure 1). Subsequently, the decompositions are performed (in DECOMP) by calling ILUDEC at all computational levels, $k = 1, 2, \ldots, l$.

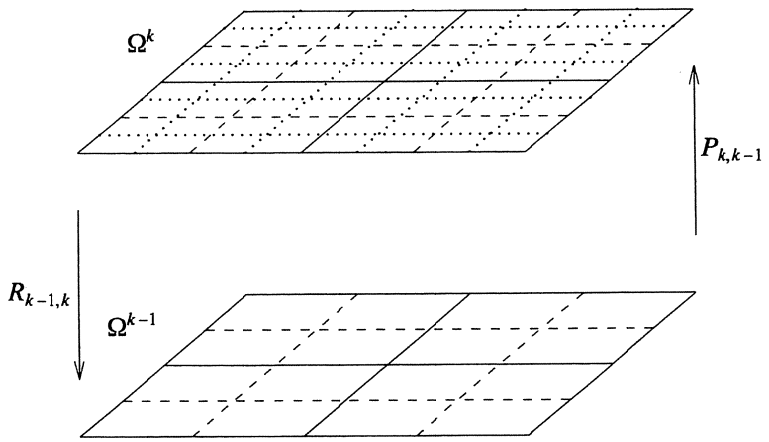


FIGURE 1. Restriction operator $R_{k-1,k}$ and prolongation operator $P_{k,k-1}$.

At the beginning of an MG-iteration cycle, u_l contains the current approximation, and r_l the corresponding residual. We take $u_l = 0$ and $r_l = f$ if no initial estimate is available. The MGD1M iteration process is

accomplished by routine CYCLES as follows:

```
C       The iteration process

        DO 100 N = 1, MAXIT
          CALL RESTRI( F, R, L-1 )        f_{l-1} = R_{l-1,l} r_l
          DO 10 K = L-2, 1, -1
            CALL RESTRI( F, F, K )        f_k = R_{k,k+1} f_{k+1}
10        CONTINUE

          CALL SOLVE( U, F, 1 )           u_1 = (L_1 U_1)^{-1} f_1

          DO 20 K = 2, L-2
            CALL PROLON( U, U, K )        u_k = P_{k,k-1} u_{k-1}
            CALL CTUPF( V, U, F, K )      v_k = C_k u_k + f_k
            CALL SOLVE( U, V, K )         u_k = (L_k U_k)^{-1} v_k
20        CONTINUE

          CALL PROLON( R, U, L )          r_l = P_{l,l-1} u_{l-1}

          DO 30 J = 1, NXF * NYF
            R( J ) = R( J ) + U( J )      r_l = r_l + u_l
30        CONTINUE

          CALL CTUPF( V, R, F, L )        v_l = C_l r_l + f_l

          CALL SOLVE( U, V, L )           u_l = (L_l U_l)^{-1} v_l

          CALL CTUMV( U, R )              r_l = C_l u_l - r_l

          RES = SNRM2( R )                ||r_l||_2

          IF( RES.LT.TOL ) GOTO 200
100     CONTINUE
200     CONTINUE
```

The purpose of the routines SOLVE, CTUPF (C Times U Plus F) and (C Times U Minus V) will be clear from the above scheme.

3. Performance analysis tools in a multigrid environment

To analyze the vector and parallel performance, we make use of the following Cray performance analysis tools[5]:

PERFTRACE - gives statistics on computer performance for individual routines within a program

ATEXPERT - focuses on timing each parallel region identified by the Autotasking system. Times for starting, stopping, and performing the parallel work in parallel regions are measured. ATEXPERT also measures the time to perform the work with a single processor (called unitasked time) and the time between parallel regions.

In our example, on the coarsest grid we have 5×5 grid points and on the finest grid 513×513 grid points for experiments with $l = 8$. Obviously, the amount of work strongly differs per level. Results of PERFTRACE and ATEXPERT are blurred by this phenomenon. Beside using PERFTRACE a control was built in to measure the Mflops on each grid for an individual routine. This implies that the PERFTRACE report may differ from our measured Mflop rate. Moreover, PERFTRACE will not work correctly with multitasked codes. To help ATEXPERT predict more reliable speedups, on the highest level we sometimes call an alternative code which does not differ from the original code except the name of the subroutine. As a consequence, ATEXPERT reports on both routines, the one called on the coarser grids and the one called on the

finest grid. The latter will give a much better prediction for the parallel performance than the one obtained for all levels together.

In Section 4 we will show PERFTRACE reports, one of MGD1V and one of MGD1M. Output achieved by ATEXPERT can be found in Section 5. Although both PERFTRACE and ATEXPERT do not perform optimally in a multigrid environment, we consider them as useful tools.

4. VECTORIZATION

Vectorization and parallelization are closely related. It is worth noting that for the sake of parallelization some changes have been made that have a negative effect on the vector performance, especially on the coarser grids. However, since most time is spent on the finest grid, a performance improvement on that level outweighs a loss of performance on coarser grids.

4.1. To a better vector performance

In many routines the one-dimensional arrays have been replaced by two-dimensional arrays, and consequently single loops have been replaced by loop nests. By doing so, the readability of the routines has been improved considerably and secondly, parallel processing can be obtained without using directives. A list of the most significant changes to improve the vector performance follows:

- Stripmining has been removed. On the Cyber 205 stripmining was necessary for vectors longer than 65535 elements.
- The ratio memory access/number of floating point operations has been changed with respect to the original version. We attempted to minimize the number of temporary stores. In a straightforward implementation, the left and right boundaries and the inner area have to be treated differently. The original loop structure in MGD1V is

```
DO
  Update left boundary
  Apply first update to inner area
CONTINUE

DO
  Apply second update to inner area
  Update right boundary
CONTINUE
```

  This approach minimizes the number of page faults on a virtual memory machine like the Cyber 205. In the first DO-loop the update of the inner area and the left boundary is comparable. The same is true for the second update of the inner area and the update on the right boundary. For the Cray Y-MP we changed this piece of code into

```
DO
  Update left boundary
  Apply first and second update to inner area
  Update right boundary
CONTINUE
```

  An important additional effect is that the overhead for autotasking decreases.
- BLAS routines have been introduced. For MGD1M this implies that calls to the routines VL2NOR and COPY have been replaced by calls to SNRM2 and SCOPY, respectively.
- The routines FOURTH and ZERO are no longer called. The first one multiplies a vector by a factor 0.25, the second one initializes a vector with the value 0. Originally, these routines were developed to force the program to operate on (long) one-dimensional vectors instead of (short) two-dimensional vectors.

4.2. Comments on the vector performance

Table 1 reviews the Perfview report of the original MGD1V code measured for eight levels, with a finest grid of 513 × 513. The routines SOLVE and ILUDEC together appear to be responsible for more than 65% of the total CPU time. Below we will describe both routines briefly.

Name	Called	Time	Avg Tim	EX %	ACM %	Mmems	Mflops
SOLVE	48	5.67E-01	1.18E-02	50.8	50.8	78.0	52.0
ILUDEC	8	1.69E-01	2.11E-02	15.1	65.9	66.5	60.3
GALERK	7	1.23E-01	1.76E-02	11.0	76.9	182.3	121.5
CTUPF	42	8.66E-02	2.06E-03	7.8	84.7	243.2	145.8
CTUMV	6	5.24E-02	8.73E-03	4.7	89.4	361.3	180.6
PROLON	42	3.53E-02	8.40E-04	3.2	92.5	164.3	89.5
RESTRI	42	2.59E-02	6.17E-04	2.3	94.9	162.8	142.3
CYCLES	2	1.54E-02	7.72E-03	1.4	96.2	308.8	102.3
MATRHS	1	1.40E-02	1.40E-02	1.3	97.5	150.9	1.0
VL2NOR	8	1.35E-02	1.69E-03	1.2	98.7	156.1	311.6
ZERO	8	5.79E-03	7.24E-04	0.5	99.2	152.2	0.0
FOURTH	7	5.46E-03	7.80E-04	0.5	99.7	226.7	113.3
COPY	1	1.73E-03	1.73E-03	0.2	99.9	304.3	0.0
MGD1V	2	9.11E-04	4.55E-04	0.1	99.9	12.1	0.1
EXAMPL	1	4.12E-04	4.12E-04	0.0	100.0	16.7	0.2
P01AAF	1	6.55E-05	6.55E-05	0.0	100.0	13.0	0.2
PREPAR	2	2.63E-05	1.32E-05	0.0	100.0	19.3	0.3
DECOMP	1	2.35E-05	2.35E-05	0.0	100.0	57.7	1.4
RAP	1	1.88E-05	1.88E-05	0.0	100.0	47.9	0.1
TIMING	10	1.27E-05	1.27E-06	0.0	100.0	3.9	3.6
X04ABF	7	1.18E-06	1.68E-07	0.0	100.0	23.8	0.0
X04AAF	6	9.84E-07	1.64E-07	0.0	100.0	24.4	0.0
Totals	253	1.12E+00		100.0	100.0	125.0	80.6

TABLE 1. Perfview report of MGD1V for 8 levels.

A 7-point discretization operator (illustrated in Figure 2)

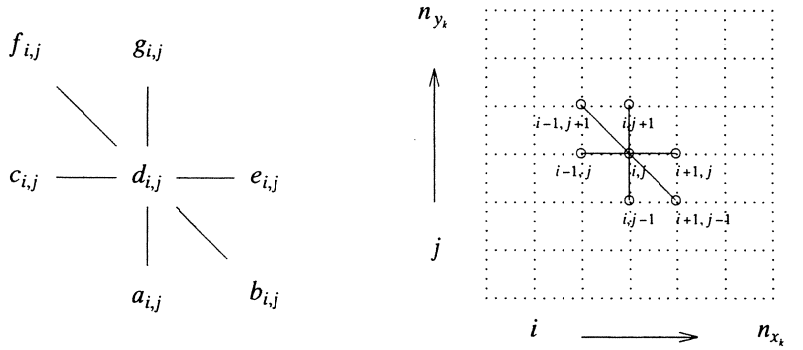


FIGURE 2.

at an interior node (i, j) in a 2-D grid gives the following linear equation

$$a_{i,j}u_{i,j-1} + b_{i,j}u_{i+1,j-1} + c_{i,j}u_{i-1,j} + d_{i,j}u_{i,j} + e_{i,j}u_{i+1,j} + f_{i,j}u_{i-1,j+1} + g_{i,j}u_{i,j+1} = F_{i,j}. \qquad (4.2.1)$$

At each level k, the diagonal matrix A_k with diagonals ($\{ a_{i,j} \}$,$\{ b_{i,j} \}$,$\{ c_{i,j} \}$,$\{ d_{i,j} \}$,$\{ e_{i,j} \}$,$\{ f_{i,j} \}$,$\{ g_{i,j} \}$),

$i = 1, \cdots, n_{x_k}, j = 1, \cdots, n_{y_k}$ has to be decomposed according to equation (2.2)

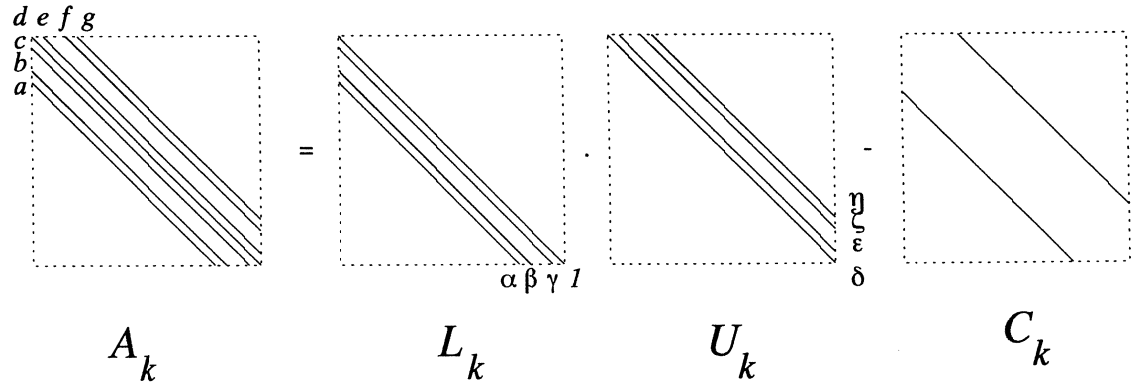


FIGURE 3. The Incomplete LU-decomposition.

The elements of the lower matrix L_k and the upper matrix U_k (see Figure 3) can be computed as described in Sonneveld et al.[10]. For L_k we obtain

$$\begin{aligned} \alpha_{i,j} &= a_{i,j}/\delta_{i,j-1}, \\ \beta_{i,j} &= (b_{i,j} - \alpha_{i,j}\,\varepsilon_{i,j-1})/\delta_{i+1,j-1}, \\ \gamma_{i,j} &= (c_{i,j} - \alpha_{i,j}\,\zeta_{i,j-1})/\delta_{i-1,j}, \end{aligned} \tag{4.2.2a}$$

and for U_k

$$\begin{aligned} \delta_{i,j} &= d_{i,j} - \gamma_{i,j}\,\varepsilon_{i-1,j} - \beta_{i,j}\,\zeta_{i+1,j-1} - \alpha_{i,j}\,\eta_{i,j-1}, \\ \varepsilon_{i,j} &= e_{i,j} - \beta_{i,j}\,\eta_{i+1,j-1}, \\ \zeta_{i,j} &= f_{i,j} - \gamma_{i,j}\,\eta_{i-1,j}, \\ \eta_{i,j} &= g_{i,j}, \end{aligned} \tag{4.2.2b}$$

for $i = 1, \cdots, n_{x_k}, j = 1, \cdots, n_{y_k}$. A straightforward implementation of (4.2.2a-b) leads to recursive relations that do not vectorize well. Below, we show an ordering scheme which leads to exactly the same decomposition and which can be computed data independently. First we focus on the back substitution which is carried out by the routine SOLVE.

In routine SOLVE the solution of the system $L_k\,U_k\,x = q$ is obtained by back substitution

$$L_k\,y = q,$$

$$U_k\,x = y.$$

For these systems the following equations have to be solved:

$$y_{i,j} = q_{i,j} - \gamma_{i,j}\,y_{i-1,j} - \beta_{i,j}\,y_{i+1,j-1} - \alpha_{i,j}\,y_{i,j-1}, \tag{4.2.3a}$$

$$x_{i,j} = (y_{i,j} - \varepsilon_{i,j}\,x_{i+1,j} - \zeta_{i,j}\,x_{i-1,j+1} - \eta_{i,j}\,x_{i,j+1})\,/\,\delta_{i,j}, \tag{4.2.3b}$$

for $i = 1, \cdots, n_{x_k}, j = 1, \cdots, n_{y_k}$. For the computation of the element $y_{i,j}$ of (4.2.3a) updated values of its neighbors $y_{i-1,j}$, $y_{i+1,j-1}$ and $y_{i,j-1}$ are required. This coupling which is illustrated in Figure 4 leads to recurrence relations, when the rows are computed sequentially, for each point $y_{i,j}$ depends on its neighbor $y_{i-1,j}$ which belongs to the same row j. The second relation (4.2.3b) leads to a similar coupling (cf. Figure 4).

Coupling (4.2.3a) Coupling (4.2.3b)

FIGURE 4. Coupling molecules for relations (4.2.3a-b) .

For an optimal implementation we used the highly efficient routines

FOLR - solving first order linear recurrences
FOLR2 - solving first order linear recurrences without overwriting operands

both from the Cray SCIentific LIBrary[3]. On the finest grid 55 Mflops has been measured (see Table 2), a performance gain of 17% compared to MGD1V.

Ashcraft and Grimes[1], however, describe an algorithm where the computations are reordered allowing them to be vectorized. The nodes are ordered in the natural ordering and the vector implementation is algebraically equivalent to the scalar recursive implementation based on FOLR and FOLR2.
For the ordering of Ashcraft and Grimes we define the following set of nodes on a 2D-grid of $n_x \times n_y$ points. Let S_m be the set of nodes

$$S_m = \{ (i,j) \mid i + 2j = m \}, \qquad m=3, \cdots, n_x+2(n_y-1) \tag{4.2.4}$$

then for the coupling (4.2.3a) (cf. Figure 4), the elements of S_m are data independent. The dashed lines in Figure 5 display the sets S_m, $m=5, \cdots, n_x+2n_y-4$. We see that S_m only depends on sets S_{m-1} and S_{m-2}. Thus, once the computations have been done for the nodes in S_{m-1}, all computations for the nodes in S_m may be done, simultaneously and can therefore be vectorized. When the elements have been coupled corresponding to the right molecule(4.2.3b) in Figure 4, we can use the same set of nodes S_m as described in (4.2.4). However, in this case we start at the upper right corner (n_x,n_y) and end at the lower left corner (1,1).

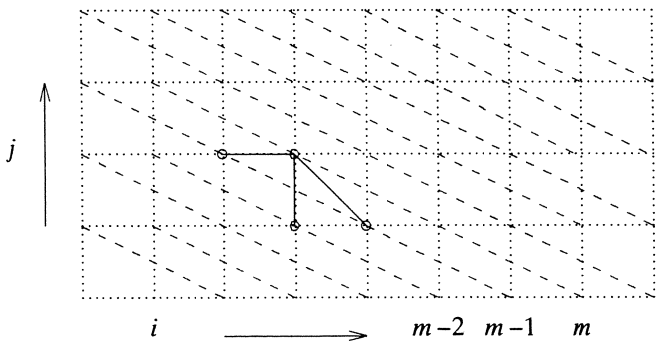


FIGURE 5. Knight's move ordering of Ashcraft and Grimes .

Our first implementation of the so-called knight's move ordering was based on a two-dimensional approach. In this implementation boundary nodes were considered separately, since they have nonexistent neighbors. Moreover, not every set S_m has exactly two boundary nodes. Unfortunately, such a well-behaved implementation with a minimum of floating point operations leads to a performance decrease (cf. Table 2, column 4). Table 2, column 5 results on a one-dimensional approach which ignores nonexistent neighbors. The values of L_k or U_k whose subscripts are outside the range $(1 : n_x \times n_y)$ are replaced by zero.

Number of Grid points	MGD1V	MGD1M		
	Vector	FOLR(2)	2-dimensions	1-dimension
513 x 513	54	55	44	155
257 x 257	51	52	43	127
129 x 129	45	48	38	93
65 x 65	37	40	28	57
33 x 33	29	32	18	31
17 x 17	24	21	10	16
9 x 9	17	12	5	9
5 x 5	11	7	2	5

TABLE 2. Results for SOLVE in Mflops.

It can easily be verified that the ILU-decomposition (4.2.2a-b) can subsequently be performed analogously, using the same set of nodes. For a fixed m we start to compute the α's. Next, the β's and γ's can be computed concurrently. Then the δ's can be calculated with the values of α , β and γ which belong to set S_m. Finally, we obtain the values of η and ζ in parallel, too. Notice that for the knight's move approach the average vector length is

$$\frac{n_x\, n_y}{n_x + 2\,(n_y - 1)} \approx \frac{n_x}{3},$$

which is about one third of the original vector length in case of $n_x = n_y$.This implies that for small grid sizes the effect of reordering is small, or, even negative.

Grid points	MGD1V	MGD1M
513 x 513	38	153
257 x 257	37	130
129 x 129	36	100
65 x 65	34	65
33 x 33	32	35
17 x 17	27	19
9 x 9	19	10
5 x 5	13	6

TABLE 3. Results for ILUDEC in Mflops.

Finally, we present the Perfview report of the most-time consuming routines of MGD1M in Table 4. The algorithm is no longer recursive and the average Mflop rate has become 154 Mflops. Remember that Perfview reports on the *vector* and not on the *parallel* performance. So, the final Mflop rate will become much higher when parts of the program can be performed in parallel.

Name	Called	Time	Avg Tim	EX %	ACM %	Mmems	Mflops
SOLVE	48	1.97E-01	4.11E-03	39.7	39.7	181.6	141.8
CTUPF	42	6.19E-02	1.47E-03	12.5	52.2	271.9	203.8
CTUMV	6	5.44E-02	9.07E-03	11.0	63.2	346.3	172.9
ILUDEC	8	4.50E-02	5.62E-03	9.1	72.3	218.4	157.1
GALERK	7	4.26E-02	6.08E-03	8.6	80.9	214.2	154.9
OTHERS	118	9.51E-02		19.1	100.0		
Totals	229	4.96E-01		100.0	100.0	218.0	153.8

TABLE 4. Perfview report of MGD1M for 8 levels.

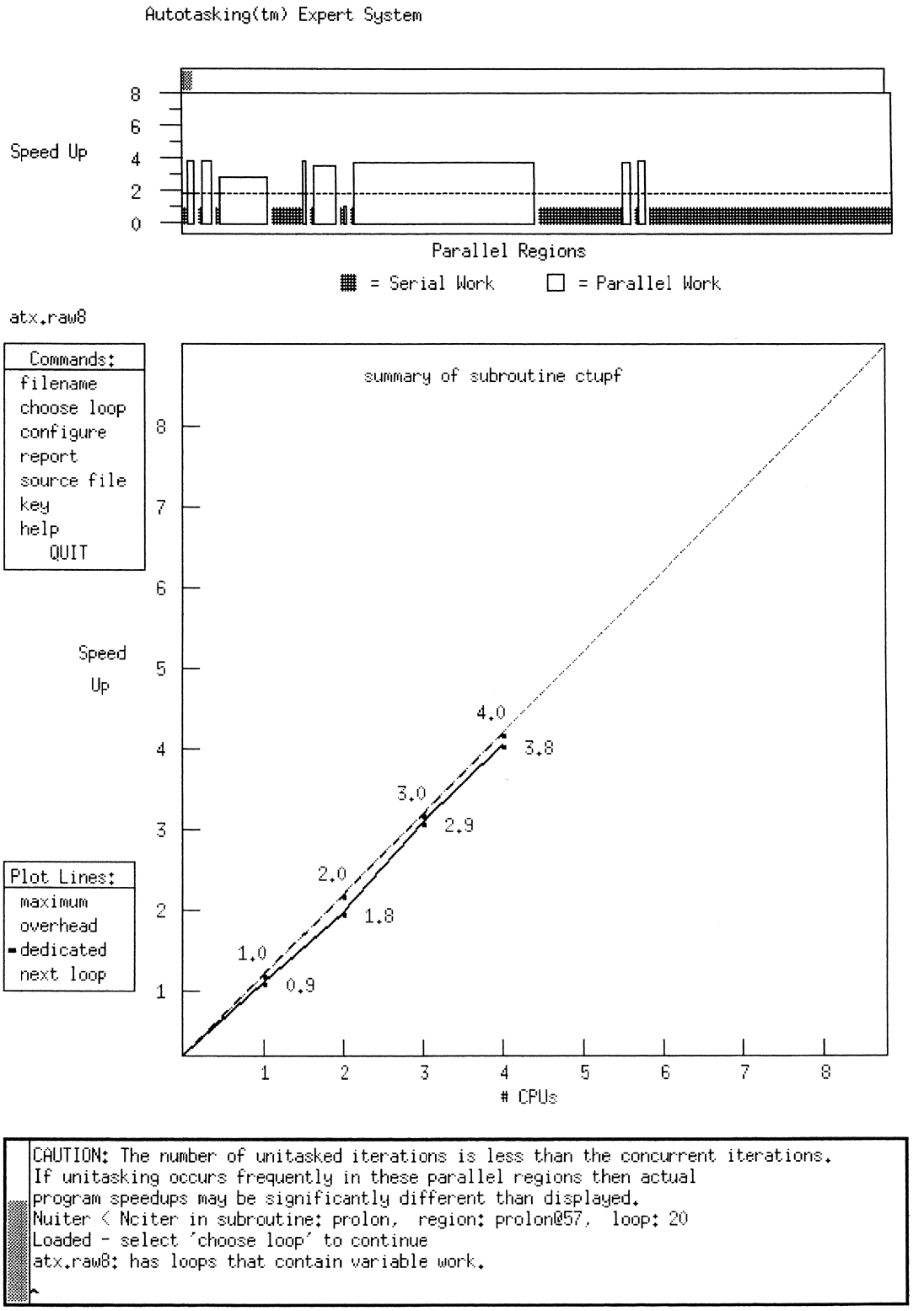


FIGURE 6. Parallel speedup for CTUPF as predicted by ATEXPERT.

5. PARALLELISM

The original code MGD1V operates on (long) one-dimensional vectors without any directives or special compiler options. The amount of parallelism is small. In Section 5.1, we discuss how we apply autotasking. An example of macrotasking is illuminated in Section 5.2.

5.1. Autotasking

A simple way to introduce parallelism is to add one of the following CMIC$ directives[2]

- CMIC$ DO PARALLEL .... SHARED(..) PRIVATE (..)
- CMIC$ DO ALL AUTOSCOPE

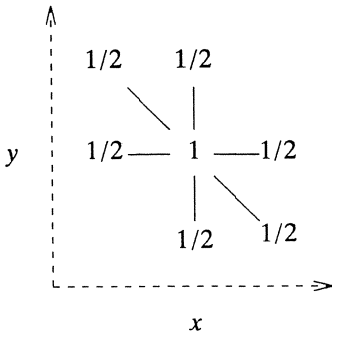
If the number of floating point operations within a loop is large enough then it pays to perform the operations simultaneously. The use of the first directive requires some additional work of the programmer. Insertion of the second directive, however, forces the compiler to autotask. Obviously, the use of both directives results in the same speedup. Because of its simplicity we used the latter one. Another solution to extend the parallelism is to treat the problem two-dimensionally instead of one-dimensionally, wherever possible. In that case the compiler can decide to perform the outer loop in parallel.

Megaflops measured on a dedicated Cray Y-MP4/464							
Routine	Number of Grid points	MGD1V	MGD1M				
		Vector	Vector	$p = 1$	$p = 2$	$p = 3$	$p = 4$
GALERK	257 x 257	125	182	182	361	480	692
	129 x 129	115	172	172	340	502	648
	65 x 65	99	156	155	303	441	576
	33 x 33	79	136	132	250	356	424
	17 x 17	47	88	84	149	195	238
	9 x 9	20	52	47	76	94	108
	5 x 5	8	30	29	29	29	29
CTUPF	513 x 513	165	210	218	432	640	846
	257 x 257	164	213	215	425	628	815
	129 x 129	161	209	209	403	584	724
	65 x 65	159	206	194	356	489	580
	33 x 33	150	184	158	252	294	328
	17 x 17	121	131	98	122	132	136
	9 x 9	67	63	41	40	40	40
CTUMV	513 x 513	183	194	176	350	505	623
RESTRI	257 x 257	153	154	153	301	444	577
	129 x 129	137	139	136	264	386	498
	65 x 65	112	115	111	209	296	372
	33 x 33	77	79	75	132	177	207
	17 x 17	44	46	42	66	80	88
	9 x 9	23	23	23	23	23	23
	5 x 5	10	11	10	10	10	10
PROLON	257 x 257	139	162	160	316	466	615
	129 x 129	127	150	147	288	420	547
	65 x 65	103	130	125	237	338	426
	33 x 33	84	89	84	148	197	238
	17 x 17	41	52	46	73	92	100
	9 x 9	19	26	26	26	26	26
	5 x 5	9	12	11	11	11	11

TABLE 5. Results for CTUPF, CTUMV, RESTRI, PROLON, and GALERK in Mflops.

As an illustration we include a picture obtained by ATEXPERT (see Figure 6). It shows us the predicted parallel speedup for the routine CTUPF. Table 5 shows the Megaflops measured on a nearly dedicated machine. The results in the first two columns have been achieved for subroutines compiled with '-Zv'. The other results were obtained by compiling with '-Zp' and taking NCPUS=p. The results indicate that the grid size must be large enough in order to get a positive effect on autotasking.

We concentrate on two highly parallel routines: PROLON and RESTRI. PROLON computes the prolongation A_k from grid A_{k-1} and RESTRI calculates the restriction A_{k-1} from grid A_k in accordance with the molecule



When, for a single grid point, all contributions of the prolongation, or, the restriction are added in one computational statement, i.e., it is updated once per grid point then the highest speedup is obtained. The implementation of PROLON and RESTRI turns out to be highly parallel. The parallel speedup for these routines achieved for grids of 257 x 257 and 129 x 129 points are shown in Table 6.

Parallel Speedup on p = 2, 3, 4 processors.			
ROUTINE	S_2	S_3	S_4
RESTRI(257)	1.95	2.88	3.75
RESTRI(129)	1.90	2.78	3.58
PROLON(257)	1.95	2.88	3.80
PROLON(129)	1.92	2.80	3.65

TABLE 6. Parallel speedup compared to vector performance .

The highest speedup factor (vector as well as parallel) is achieved by Galerkin. In [9] Lioen describes how for the Galerkin approximations the amount of work per grid point can be reduced from 191(cf. Wesseling[11]) to 74 floating point operations. Moreover, the strongly improved implementation appears to be highly parallel. On four processors we obtain a parallel speedup of 3.80 for $n_x = n_y = 257$.

5.2. Macrotasking

The incomplete LU-decomposition on two different grids are data independent. This implies that these decompositions can be done in parallel. If we define

$$\tau_k = \text{time to decompose } A_k \tag{5.3.1a}$$

$$T_k = \text{time to decompose } A_1, \cdots, A_k \tag{5.3.1b}$$

then we obtain

$$T_{l-1} = \sum_{k=1}^{l-1} \tau_k \approx \left(\cdots + \frac{1}{16}\tau_{l-1} + \frac{1}{4}\tau_{l-1} + \tau_{l-1} \right) \approx 1\frac{5}{16}\tau_{l-1} \tag{5.3.2a}$$

$$T_l = \sum_{k=1}^{l} \tau_k = T_{l-1} + \tau_l \approx 5\frac{5}{16}\tau_{l-1}. \tag{5.3.2b}$$

From (5.3.2a-b) we conclude that $T_{l-1} \ll \tau_l$. Thus, the decomposition of the first $l-1$ levels can be performed simultaneously with the decomposition of level l. The theoretical speedup is

$$S_2 \approx \frac{T_l}{\tau_l} \approx \frac{5\frac{5}{16}\tau_{l-1}}{4\tau_{l-1}} = 1.33. \tag{5.3.3}$$

6. CONCLUSIONS AND REMARKS

As a test problem we solve the Poisson equation on the unit square with Dirichlet boundary conditions, zero initial estimates and the right-hand-side constructed according to the exact solution

$$x\,(1-x) + y\,(1-y). \tag{6.1}$$

This problem was also used to measure the performance of MGD1V on the Cyber 205 [12]. The convergence of the tested multigrid method is so rapid that only a few multigrid iterations are needed. After six iterations the Euclidian norm of the residu is less than 10^{-9}. The code uses a fixed multigrid strategy. Therefore the particular choice of the test problem is not important. Clearly, the required number of multigrid iterations depends on the problem. The problem is solved using 7 as well as 8 levels, with a coarsest grid of 5 × 5 grid points.

In Table 7, we present the wall-clock time measured in seconds. We make a distinction between the vector and the parallel performance on one processor. The first one does not apply macrotasking. It has been compiled with the '-Zv' option. The latter contains auto- and macrotasking directives, and it has been compiled with '-Zp'. Due to the parallel overhead the latter will always produce slower execution times. For that reason the execution time of the vector run is included in the speedup definitions (6.2) and (6.3). The vector speedup is defined by

$$S_{vector} = \frac{\textit{Execution time of } \text{MGD1V}}{\textit{Execution time of } \text{vector-MGD1M}} \tag{6.2}$$

both measured on a dedicated Cray Y-MP4. The parallel speedup is denoted by

$$S_p = \frac{\textit{Execution time of } \text{vector-MGD1M } \textit{on 1 processor}}{\textit{Execution time of } \text{MGD1M } \textit{on p processors}}. \tag{6.3}$$

The total speedup in wall clock time can be expressed as the product of

$$S_{vector} \times S_4.$$

According to [12], we distinguish three phases in the multigrid solution process, viz., the computation of the Galerkin approximations, the ILU decomposition, and finally, the multigrid iteration process. At present, the values listed in Table 7 are not as accurate as we desire. Although they were achieved on a 'dedicated' machine, we obtain at most 86% when running on 4 processors. For 'Decompose' we measured a parallel speedup of 1.41, which exceeds the theoretical speedup of 1.33 (cf. (5.3.3)). This is not as surprising as it seems to be. In (5.3.3) we do not take into account the vector performance, which increases with the grid size.

In spite of the 'nondedicateness' of the Cray Y-MP, we will draw some cautious conclusions. The main reduction in wall-clock time arises from vectorization and not from parallelization caused by the knight's move ordering applied in both the decomposition and the cycling process. As a consequence, the multigrid code MGD1M can be completely vectorized. This new ordering can not easily be parallelized, partly because the average vector length has been reduced to one third of the original vector length. Therefore, parallelization must be introduced at a higher level, for instance, by applying a twisted factorization as proposed by Joubert et al.[8].

	MGD1V	MGD1M					
	Vector	Vector	S_{vector}	$p = 1$	$p = 2$	$p = 3$	$p = 4$
Galerkin	.131	.043		.042	.021	.014	.013
S_p, $p = 2, 3, 4$			3.05		2.00	2.96	3.16
Decompose	.169	.045		.046	.032	.032	.032
S_p, $p = 2, 3, 4$			3.77		1.41	1.41	1.41
MG-cycles	.774	.380		.380	.295	.268	.255
S_p, $p = 2, 3, 4$			2.03		1.29	1.42	1.49
Total	1.074	.468		.468	.348	.314	.300
S_p, $p = 2, 3, 4$			2.29		1.34	1.49	1.56

TABLE 7. Wall-Clock time in seconds for 8 Levels and Speedups.

ACKNOWLEDGEMENT

The author would like to thank Paul de Zeeuw for his clear introduction in the multigrid software. His explanation enabled her to optimize the multigrid code so efficiently, and to do this within a very short period. The author is also very grateful to the people of SARA for helping her to run the timing programs on a dedicated machine.

Financial support for this work was provided by Cray Research, Inc. under a 1991 University Research & Development Grant for the project 'Numerical Multigrid Software' for the Cray Y-MP4. This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF) for use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO).

REFERENCES

1. C. C. ASHCRAFT and R. G. GRIMES (1988). On vectorizing incomplete factorization and SSOR preconditioners, *SIAM J. Sci. Stat. Comput.*, Vol. 9, No 1, pp. 122-151.
2. CRAY CF77 COMPILING SYSTEM (1990). Volume 4: Parallel Processing Guide, Publication SG-3074 4.0.
3. CRAY LIBRARY REFERENCE MANUAL (1989). Volume 3: UNICOS Math and Scientific Library reference Manual, Publication SR-2081 5.0.
4. CRAY MULTITASKING PROGRAMMER'S MANUAL (1989). Publication SR-0222 F.
5. CRAY REFERENCE MANUAL (1991). UNICOS Performance Utilities Reference Manual, Publication SR-2040 6.0.
6. P.W. HEMKER, R. KETTLER, P. WESSELING and P.M. DE ZEEUW (1983) Multigrid Methods: Development of Fast Solvers, *Appl. Math. Comp.*, 13, pp. 311-326.
7. P.W. HEMKER and P.M. DE ZEEUW (1985). Some implementations of multigrid linear systems solvers, in: D.J. Paddon and H. Holstein, Eds., *Multigrid Methods for Integral and Differential Equations*, Inst. Math. Appl. Conf. Ser. New Ser. (Oxford Univ. Press, New York), pp. 85-116.
8. G.R. JOUBERT and E. CLOETE (1985). The Solution of Tridiagonal Linear Systems with an MIMD Parallel Computer, *ZAMM Z. Angew. Math. u. Mech.*, Vol. 65, No 4, pp. T383-385.
9. W.M. LIOEN (to appear). *An optimally efficient algorithm for Galerkin Coarse-grid approximation*, C WI.
10. P. SONNEVELD, P. WESSELING and P.M. DE ZEEUW (1985). Multigrid and Conjugate Gradient Methods as convergence acceleration techniques, in: D.J. Paddon and H. Holstein, Eds., *Multigrid Methods for Integral and Differential Equations* , Inst. Math. Appl. Conf. Ser. New Ser. (Oxford Univ. Press, New York), pp. 117-167.

11. P. WESSELING (1982). A robust and efficient multigrid method, in: W. Hackbusch and U. Trottenberg, Eds., *Lecture Notes in Mathematics*, 960 (Springer, Berlin), pp. 614-630.
12. P.M. DE ZEEUW (1986). NUMVEC FORTRAN library manual. Chapter: Elliptic PDEs, Routine: MGD1V and MGD5V; CWI Report NM-R8624.
13. P.M. DE ZEEUW and E.J. VAN ASSELT (1985). The convergence rate of multi-level algorithms applied to the convection-diffusion equation, *SIAM J. Sci. Statist. Comput.* 6 (2), pp. 492-503.