Parallel Implementations Of Multigrid Methods

Linda Stals

In D. Stewart, H. Gardner and D. Singleton, editors
CTAC93, Proceedings of the Sixth Biennial Conference On Computational Techniques and Applications, Canberra, Australia.
Pages 437-443. World Scientific, 1993.

1. Introduction

The use of multigrid methods on serial machines has proven to be an effective way of solving a wide range of problems. But how well do they perform on parallel machines? The need to sequentially move down the grid levels and problems with idle processors might give one the impression that multigrid methods are inherently serial. The aim of this paper is to show that this is not true, in fact even the standard multigrid methods can be used as fairly efficient parallel algorithms.

The paper is divided into two sections. The first section introduces the notation by defining the standard multigrid method, MG, and the full multigrid method, FMG. The second section looks at the parallel implementation of the standard multigrid methods. It discusses how the methods can be implemented on a parallel machine and presents some results that I have obtained on the AP1000.

2. Standard Multigrid Methods

Let us consider an elliptic PDE of the form

$$\begin{aligned} Au &= f \quad in\, \Omega \\ u &= g \quad on\, \partial\Omega \end{aligned} \tag{2.1}$$

After applying some discretization method such as the finite element or finite difference methods we are left with the discrete problem

$$A_h u_h = f_h \tag{2.2}$$

The classical method of solving 2.2 is to use some iterative method such as the Jacobi or Gauss-Seidel methods. Iterative methods typically have difficulty with removing the low frequency components of the error and this tends to slow the convergence after several iterations. Multigrid methods overcome this by applying the relaxation method to the error on a coarse grid. Such an approach is effective because low freqency components of the error become high frequency components when the error is moved to a coarse grid.

To define the multigrid methods we firstly need a sequence of grids

$$\mathcal{M}_1 \subset \mathcal{M}_2 \subset \cdots \subset \mathcal{M}_n \tag{2.3}$$

and the following set of operations.

Projection: $P_m^{m-1} : \mathcal{M}_m \rightarrow \mathcal{M}_{m-1}$
Interpolation: $I_{m-1}^{m} : \mathcal{M}_{m-1} \rightarrow \mathcal{M}_m$
Relaxation: $R_m : \mathcal{M}_m \rightarrow \mathcal{M}_m$

The actual algorithm for the standard multigrid method is;

- $MG^m(p, \mu_1, \mu_2, f^m, v^m)$
  1. repeat μ_1 times
     $v^m \leftarrow v^m + R_m(f^m - A^m v^m)$ (Pre Smoothing)
  2. if $m = 1$ goto (8)
  3. $f^{m-1} \leftarrow P_m^{m-1}(f^m - A^m v^m)$ (Projection)
  4. $v^{m-1} \leftarrow 0$
  5. repeat p times
     $v^{m-1} \leftarrow MG^{m-1}(p, \mu_1, \mu_2, f^{m-1}, v^{m-1})$ (Coarse Grid Approximation)
  6. $e^m \leftarrow I_{m-1}^m v^{m-1}$ (Interpolation)
  7. $v^m \leftarrow e^m + v^m$ (Correction)
  8. repeat μ_2 times (Post Smoothing)
     $v^m \leftarrow v^m + R_m(f^m - A^m v^m)$
  9. return v^m

The purpose of steps 3 to 6 is to find the coarse grid approximation to the equation

$$\begin{aligned} A^m e^m &= A^m u^m - A^m v^m \\ &= f^m - A^m v^m \\ &= r^m \end{aligned}$$

The power behind the multigrid methods comes from the complementary nature of its major steps. Recall that the relaxation methods (applied in the pre and post smoothing operations) are good at removing the high freqency components of the error but tend to leave the low frequency components behind. On the other hand, the grid transfer operations, projection and interpolation, work best on low frequency components.

The number of times the algorithm visits the coarse grid is controlled by p. p is generally taken to be 1 or 2. If p is 1 the method is called the V-cycle because it moves through the grids forming a V shape, see figure 1. If p is 2 it is called the W-cycle, see figure 2.

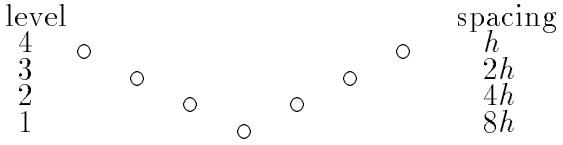

FIGURE 1. Map of the algorithm moving through the grid levels for the V-cycle

Note that in the MG algorithm we need an initial approximation for v^n. One method of finding an initial approximation is to solve the problem on a coarse grid. This leads to the definition of the full multigrid method.

- $FMG^m(r, p, \mu_1, \mu_2, f^m, v^m)$
  1. if $m = 1$ goto (7)
  2. $f^{m-1} \leftarrow P_m^{m-1}(f^m - A^m v^m)$ (Projection)
  3. $v^{m-1} \leftarrow 0$
  4. $v^{m-1} \leftarrow FMG^{m-1}(r, p, \mu_1, \mu_2, f^{m-1}, v^{m-1})$

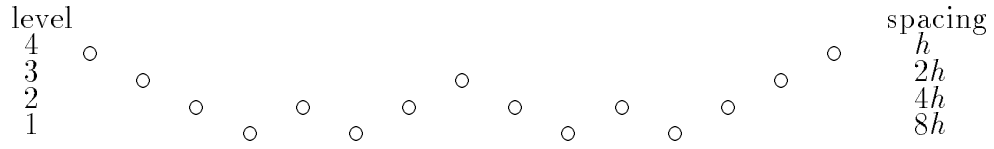

FIGURE 2. Map of the algorithm moving through the grid levels for the W-cycle

5. $e^m \leftarrow I^m_{m-1} v^{m-1}$ (Interpolation)
6. $v^m \leftarrow v^m + e^m$ (Correction)
7. repeat r times
   $v^m \leftarrow MG^m(p, \mu_1, \mu_2, f^m, v^m)$
8. return v^m

If p and r are 1 then this method is call the full multigrid V-cycle, FMV, since a map of the algorithm moving through the grids looks like a collection of v's. See figure 3

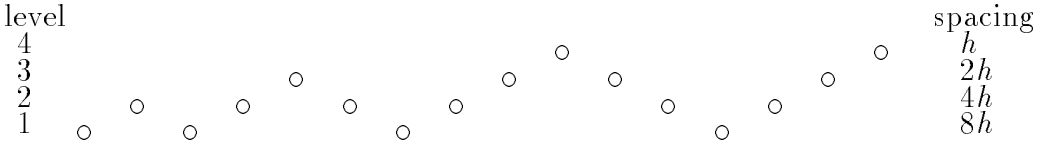

FIGURE 3. Map of the algorithm moving through the grid levels for FMV

3. Parallel Implementation Of Standard Multigrid Methods

The multigrid methods presented above can not be parallelized by performing operations on the different grid levels at once. These are recursive algorithms requiring the results from the previous grid levels. Instead, the parallelism comes from performing operations on the nodes of a particular grid in parallel.

To clarify the discussion I am going to focus on the model problem

$$-u_{xx} - u_{yy} = f(x, y) \tag{3.4}$$

on the unit square with $u = 0$ on the boundary.

Lets cover the domain with a square grid $\mathcal{M}_m = (ih^m, jh^m)$, $0 \leq i, j \leq 2^m$, $h^m = \frac{1}{2^m}$ and use the 5-point star method to approximate 3.4.

That is

$$f^m_{i,j} = \frac{4v^m_{i,j} - v^m_{i+1,j} - v^m_{i-1,j} - v^m_{i,j+1} - v^m_{i,j-1}}{(h^m)^2} \tag{3.5}$$

where $f^m_{i,j} = f(ih^m, jh^m), v^m_{i,j} = v(ih^m, jh^m)$

The first step in using the multigrid methods on a parallel machine is to divide the grid nodes up amongst the cells (or processors). Figure 4 shows how this may be done using a machine such as the AP1000 whose architecture resembles an array of

cells, see [**1**], [**2**], [**9**] and [**10**]. Methods for handling hypercube style architectures are presented in [**4**], [**6**], [**7**], [**8**] and [**11**]. If we choose a parallel relaxation scheme,

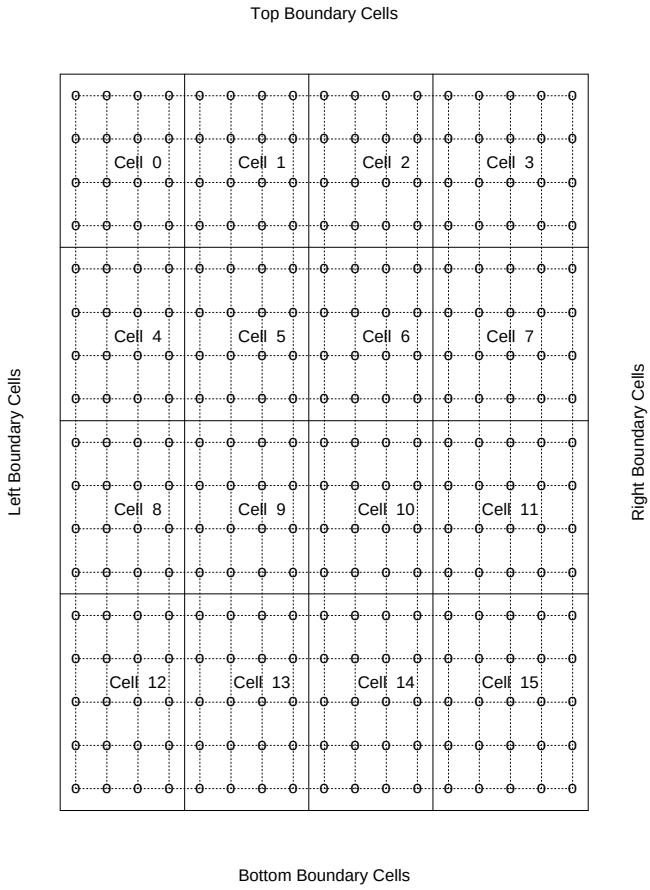

FIGURE 4. Domain decomposition. This diagram shows how a grid of level 4 is divided amongst the cells.

the pre and post smoothing operations can be performed on the grid $\mathcal{M}_m$ in parallel. For example, the Jacobi relaxation method for the model problem takes the form

$$v_{i,j}^m \leftarrow \frac{1}{4}(v_{i+1,j}^m + v_{i-1,j}^m + v_{i,j+1}^m + v_{i,j-1}^m + h^2 f_{i,j}^m) \tag{3.6}$$

which is clearly a parallel algorithm.

Further more, we can do the projection and interpolation operations in parallel. In this study I chose to use full weighting projection and bilinear interpolation;

full weighting projection:

$$\begin{aligned} f_{i,j}^{m-1} &\leftarrow \frac{1}{16}(4r_{i,j}^m + 2(r_{i+1,j}^m + r_{i-1,j}^m + r_{i,j+1}^m + r_{i,j-1}^m) \\ &+ r_{i+1,j+1}^m + r_{i-1,j+1}^m + r_{i+1,j-1}^m + r_{i-1,j-1}^m \end{aligned} \tag{3.7}$$

where $r^m = f^m - A^m v^m$.

bilinear interpolation:

$$\begin{aligned} e^m_{i,j} &\leftarrow v^{m-1}_{i,j} \\ e^m_{i+1,j} &\leftarrow \frac{1}{2}(v^{m-1}_{i,j} + v^{m-1}_{i+1,j}) \\ e^m_{i,j+1} &\leftarrow \frac{1}{2}(v^{m-1}_{i,j} + v^{m-1}_{i,j+1}) \\ e^m_{i+1,j+1} &\leftarrow \frac{1}{4}(v^{m-1}_{i,j} + v^{m-1}_{i+1,j} + v^m_{i,j+1} + v^m_{i+1,j+1}) \end{aligned} \tag{3.8}$$

The discussion so far has focused on the case where there are more grid nodes then cells, but what happens at the coarse grids? Figure 5 shows how the grids are distributed across the cells as the method moves down the grid levels. At level 1 we see that there are cells which do not have any grid nodes. These idle cells represent a loss in efficiency.

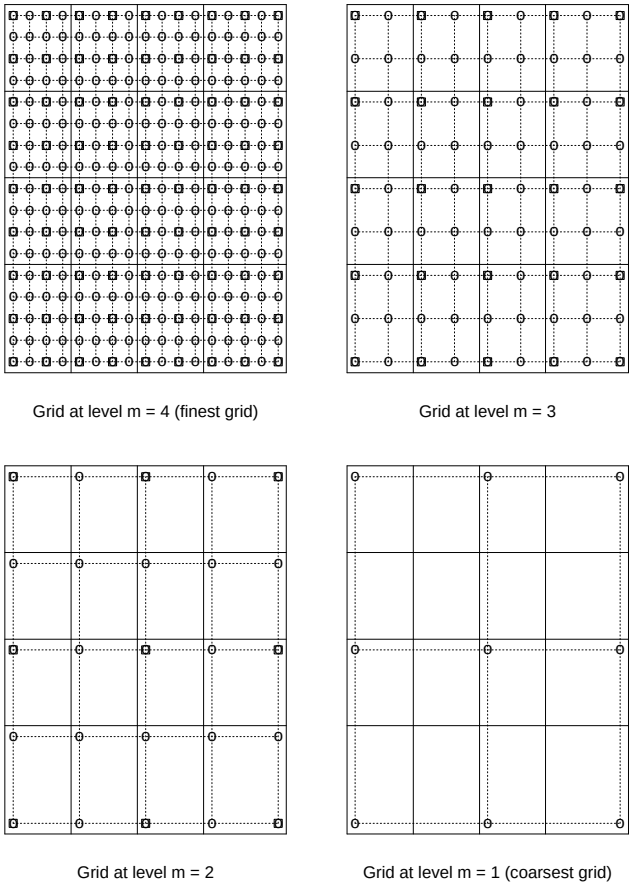

FIGURE 5. Moving down the grid levels. For m = 1, the cells which do not contain any grid points are inactive.

Despite the idle cells the multigrid methods still give a fairly high efficiency rate, because the amount of time spent in the fine grids is higher than the time spent in the coarse grids. Therefore, if the number of grid levels is high enough the loss of effieciency due to idle cells is negligible.

To back up these arguments I have some results that I have obtained by solving the model problem on the AP1000. Figures 6 and 7 show the effeciency obtained for 8, 9, 10 grid levels for the V-cycle and FMV respectively. The separate readings for the same number of cells is due to different cell configuration. For example, 4 cells can be configured as a 1×4, 2×2 or 4×1 array. The different cell configurations affect the way the grid is divided up and therefore affects the performance.

I have taken the definition of efficiency, E from [**3**]

$$E = \frac{T_1(N)}{CT_C(N)} \tag{3.9}$$

where $T_C(N)$ is the time for C cells to solve a problem of size N.

In figures 6 and 7 we see an initial increase in efficiency (see the paragraph below), however, the efficiency eventually drops off as more cells are added. As the number of cells is increased, the grid is spread more thinly, so more cells are idle for a longer time. The decrease is more marked for the FMV because more time is spent in the coarse grids.

Note that for 10 and 9 grid levels the graphs show efficiency values greater then 1. This is called super-efficiency and is a consequence of memory cache effects. Naturally, when a large grid is placed on 1 cell it uses a lot of the cell's memory. In fact, I was unable to solve the model problem on 1 cell using 10 grid levels and double precision because there was not enough memory. If the same grid is spread over more cells, say 2, then a higher proportion of the grid will sit in the cells high speed memory (or cache). Therefore, the time taken to do the computations (i.e. not including communications) for the 2 cell case may be less then half of the time taken to the computations with 1 cell. This reduction may offset any increased communication costs, so the efficiency may increase or may be greater then 1.

Another feature that figures 6 and 7 highlight is that as the number of grid levels increases so does the efficiency. Increasing the number of grid levels decreases the amount of time spent in the coarse grids. If the problem size is large enough, and if there is enough memory in the cells, then the loss in efficiency due to idle cells may be so small that it can be disregarded.

References

1. Australian National University, Department Of Computer Science. *AP1000 User's Guide*, March 30 1992.
2. Australian National University, Department Of Computer Science. *Proceedings Of The Second Fujitsu-ANU CAP Workshop*, November 1991.
3. George Almasi and Allan Gottlieb. *Highly Parallel Computing*. Benjamin/Cummings Publishing Company, 1989.
4. B. Briggs, L. Hart, S. McCormick, and D. Quinlan. Multigrid Methods On A Hypercube. In *Lecture Notes In Pure And Applied Mathematics, 110*, pages 63–83. Marcel Dekker, 1988.
5. William L Briggs. *A Multigrid Tutorial*. SIAM, 1987.
6. Tony F. Chan and Youcef Saad. Multigrid algorithms on the hypercube multiprocessor. *IEEE Transaction On Computers*, C-35(11):969–977, November 1986.
7. Tony F. Chan and Robert Schreiber. Parallel networks for multigrid algorithms architecture and complexity. *SIAM J Sci Stat Comput*, 6(3):698–711, July 1985.
8. Tony F. Chan and Ray S. Tuminaro. Design and implementation of parallel multigrid algorithms. In Steve M°Cormick, editor, *Multigrid Methods: Theory, Applications*

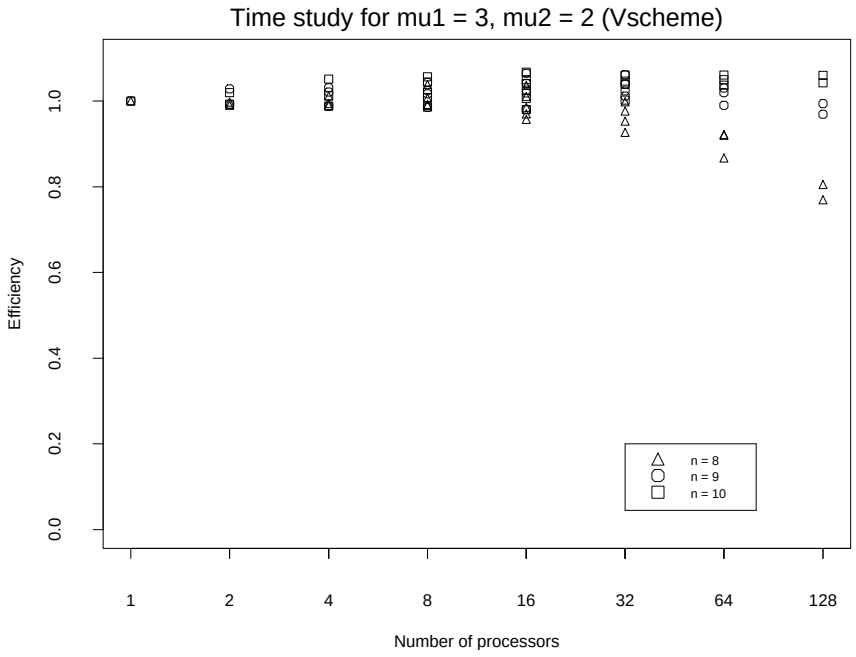

FIGURE 6. Efficiency for V-cycle using floating point.

And Supercomputing, pages 101–115. Lecture Notes In Pure And Applied Mathematics, Vol 110, 1988.

9. David Hawking. About the Fujitsu AP1000. Technical report, Department Of Computer Science, Australian National University, May 1991.
10. Hiroaki Ishihatu, Takeshi Horie, Satoshi Inano, Toshiyuki Shimizu, and Sadayuki Kato. CAP-II Architecture. Technical report, Fujitsu Laboratories Ltd., Computer-bases systems Lab., Kawasaki, 1015 Kamikodonaka, Nakahor-ku, Kawasaki, 211.
11. Oliver A. M^{c}Bryan, Paul O. Frederickson, Johannes Linden, Anton Schüller, Karl Solchenbach, Klaus Stüder, Clemens-August Thole and Ulrich Trottenberg. Multigrid methods on parallel computers - a survey of recent developments. *IMPACT Comput. Sci. Engrg*, 3:1–75, 1991.
12. Steve F. M^{c}Cormick, editor. *Multigrid Methods*. SIAM Frontiers In Applied Mathematics, 1987.

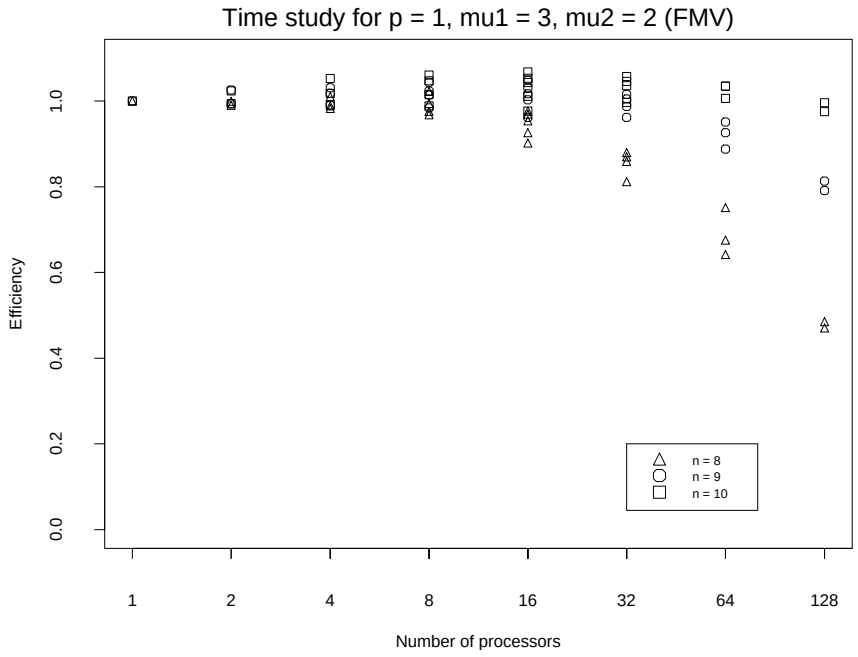

FIGURE 7. Efficiency for FMV using floating point.