ON THE STRUCTURE OF PARALLELISM IN A HIGHLY CONCURRENT PDE SOLVER

Dennis Gannon

Department of Computer Sciences
Purdue University

ABSTRACT

Abstract – This paper studies a variation of a parallel multigrid PDE solver originally due to John Van Rosendale. This paper gives a detailed analysis of the method and discusses the large scale parallel structure. It will show that the method can be viewed as a data driven "large grain" systolic structure. At a lower level the algorithm is seen to be built from grid operators that are, in turn, defined by expressions involving vector functions.

1. INTRODUCTION

Multigrid methods (studied by [Brad80], [Nico77], [VRos80] and many others) have been shown to be an effective means of solving a wide family of elliptic boundary value problems. In this paper we consider the problem of reformulating a simple, but standard multigrid method with the objective of exposing as much inherent parallelism in the underlying concepts as possible. As a consequence we derive a variation on a method first considered by Van Rosendale [GaVr82] which has several interesting properties. In particular, when the method is viewed as a serial algorithm it is inferior to the standard methods but when $2n^2$ processors are available the method can be shown to reduce the error to below that of truncation in time $0(log(n)(loglog(n) + 1))$ time as compaired to $0(log^2(n))$ for other schemes. While this result is of some theoretical interest, the most noteworthy feature of the algorithm is the structure of the parallelism.

Concurrency in the computation can be seen at three distinct levels. At the highest level the algorithm takes the form of a linear, data driven systolic array not unlike those studied by [Kung80] and others. The primary difference is that the basic item of data is a large array of real numbers. Such computations have often been called "large grain" or "macro" dataflow models. The intermediate levels of parallelism are associated with grid operations and the lowest level is in terms of vector expressions.

In Section 2 of this paper we construct the algorithm by starting with an analysis of recurrences in the "V" cycle multigrid method. Section 3 studies the numerical prperties of the method and proves a modest convergence result. Section 4 concludes with a discussion of the structure of the parallelism and indicates a model architecture for this method.

2. MULTIGRID ITERATIONS

Iterative methods for solving elliptic partial differential equations are based on using knowledge of the spectrum of an elliptic operator A to selectively reduce components of the error of an approximation to the true solution to the equation

$$Au = f.$$

Let A be defined on a two dimensional domain D and let $M_k, k = 1 \,..\, log(n)$ be a sequence of uniform 2^k by 2^k discretizations of D. Let f_k be an associated family of approximation of the function f. Using either finite element of finite difference methods we can derive an associated family of approximations A_k to A. Let $proj(\,)$ be the mapping from $H_0(M_k) \to H_0(M_{k-1})$ which, for finite difference operators is defined by the orthogonal projection (least squares approximation), and for finite element formulations is defined by the adjoint of the subspace inclusion

$$inj : H_o(M_{k-1}) \to H_0(M_k).$$

The approximations A_k are related to each other by the formula

$$projA_k inj = A_{k-1},$$

as illustrated in Figure 2.1. Using either finite element or finite difference methods, an approximation to u can be derived so that there exists a constant C such that if u_k is defined by

$$A_k u_k = f_k$$

then

$$||u - u_k||_2 \leq C(1/2)^k ||u||_2.$$

CH2146-9/85/0000/0252$01.00 © 1985 IEEE

The most trivial of all iterative schemes to approximate u_k is the Jacobi method which generates a sequence of approximations $u_k^{(s)}\ s => 1, 2, \ldots$ defined by the recurrence

$$u_k^{(s)} = u_k^{(s-1)} + \frac{1}{z_k}(f_k - A_k u_k),$$

where $u_k^{(o)}$ is an initial "guess" and z_k is the largest eigenvalue of the operator A_k. The computation of the s-*fold* iteration from $u^{(o)}$ defines an operator $Rel_k^{(s)}$ such that

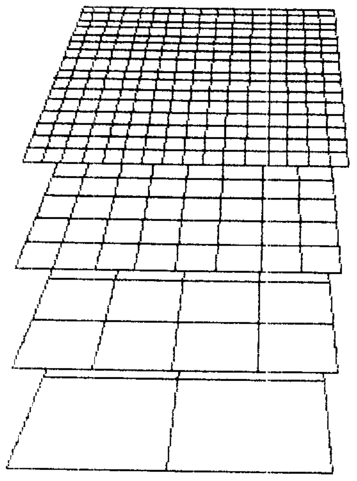
$$u_k^{(s)} = u_k^{(o)} + Rel_k^{(s)}(f_k - A_k u_k).$$

The primary measure of performance of such an iteration is the rate at which the residual vector, $r_k^{(s)} = f_k - A_k u_k^{(s)}$ can be reduced to zero. It is a simple task to show that

$$Rel_k^{(s)} = A_k^{-1}(I - (I - 1/z_k A_k)^2),$$

and that $r_k^{(s)}$ satisfies $r_k^{(s)} = (I - 1/z_k A_k)^s r_k^{(o)}$.

Consequently, if the initial guess $u_k^{(o)}$ is chosen so that $r_k^{(o)}$ contains no eigenvector component corresponding to an eigenvalue less than, say, $0.25 z_k$, then the norm of the error at iteration s satisfies $||u_k^{(s)}|| \le (0.75)^s ||u_k(o)||$. In other words, such a well chosen initial guess will permit the Jacobi iteration to converge to the solution at a rate that is independent of k. The key idea behind the multigrid method is to use the fact that A_{k-1} "nearly" approximates the first quarter of the spectrum of A_k. Therefore, if we set $u_k^{(o)}$ equal to the interpolant of u_{k-1}, the solution of $A_{k-1}u_{k-1} = f_{k-1}$, into M_k we have an initial guess that satisfies our requirements. Furthmore, we have reduced the problem to one that is on a grid of much smaller size.



$$\begin{tikzcd}
M_4 \arrow[r, "A_4"] & M_4 \arrow[d, "Proj()"] \\
M_3 \arrow[u, "Inj()"] \arrow[r, "A_3"] & M_3 \arrow[d, "Proj()"] \\
M_2 \arrow[u, "Inj()"] \arrow[r, "A_2"] & M_2 \arrow[d, "Proj()"] \\
M_1 \arrow[u, "Inj()"] \arrow[r, "A_1"] & M_1
\end{tikzcd}$$

Figure 2.1: Subgrid Relationships

A crude, but simple, variation on the standard "V" cycle formulation of this algorithm can be stated as follows:

$$U := 0;\ u_0 := 0;\ u_m := 0;\ p_m := f\ ;$$

```
repeat
    for $k$ := $m$ downto 1 do
        $p_{k-1} := proj(p_k)$ ;
    for $k$ := 1 to m do
        $u_k := inj(u_{k-1}) + Rel_k(p_k - A_k inj(u_{k-1}))$ ;
    $U := U + u_m$ ;
    $p_m = f - A_m U$ ;
until $||p_m|| <$ tolerance ;
```

where $Rel_k = Rel_k^{s_o}$ for some predefined constant s_o. The outer iteration repeats the inner approximation sequence until an acceptable tolerance has been achieved. In general, we will not use an explicit reference to the $inj()$ operator, and the second inner iteration will be written as

```
for $k$ := 1 to m do
    $u_k := u_{k-1} + Rel_k(p_k - A_k u_{k-1})$.        (2.1)
```

The body of code inside the outer iteration improves the approximation U to the solution in grid M_m at each pass. If we "unroll the loop" we see a data flow diagram illustrated in Figure 2.2.

Because of the counterposed order of the for loops it is impossible to execute this sequence in parallel in less than $0(m) = 0(log(n))$ time steps. To carry out the computation in a faster parallel time one must break this "V" cycle. Unfortunately simple reorderings of the code will not break this chain of dependences, and we must look for a fundamental reformulation of the algorithm. To do so

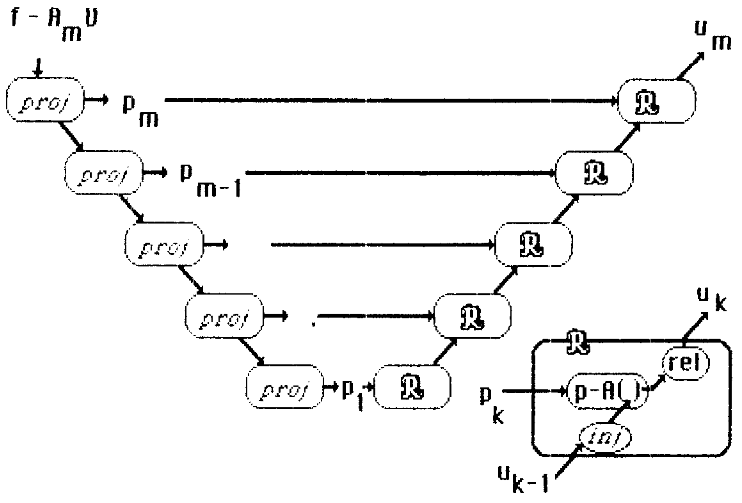


Figure 2.2: "V" Cycle Data Flow

we will rewrite the forward recurrence (2.1) as an iteration of backward recurrences involving a family of differences. Let $\delta u_k = u_k - u_{k-1}$, $v_k = p_k - p_{k-1}$, and $d_k = A_k(\delta u_k)$. From (2.1) we have

$$\begin{aligned}\delta u_k &= Rel_k(p_k - A_k u_{k-1}) \\ &= Rel_k(p_k - p_{k-1} + p_{k-1} - A_k u_k u_{k-1}) \\ &= Rel_k(v_k + p_{k-1} - A_k u_{k-1}) \\ &= Rel_k(v_k + p_{k-1} - A_{k-1}u_{k-1} + (A_{k-1} - A_k)u_{k-1}).\end{aligned}$$

Defining the residual vector $r_k = p_k - A_k u_k$, we have the pair of recurrences

$$\delta u_k = Rel_k(v_k + r_{k-1} + (A_{k-1} - A_k)u_{k-1}) \qquad (2.2)$$

$$u_k = \Sigma_{s=0}^{k}(\delta u_s) \qquad (2.3)$$

In a similar manner we can show that the residual vector satisfies the recurrence:

$$r_k = (I - A_k Rel_k)(v_k + (A_{k-1} - A_k)u_{k-1} + r_{k-1}) \quad (2.4)$$

To compute the vaue of u_m using these relations we can use the code segment shown below

```
for $k$ = 0 to m do begin (initialize)
        $\delta u_k := d_k := 0$; $v_k := p_k - p_{k-1}$ ;
        end;
for $j$ = 1 to m do begin
    $u_m := u_m + \delta u_m$ ;
    for $k = m$ downto 1 do begin
        $\delta u_k := Rel_k(v_k + d_{k-1} - A_k(\delta u_{k-1}))$ ;
        $d_k^{new} := A_k(\delta u_k)$ ;
        $v_{k+1} := v_k + d_{k-1} - A_k(\delta u_{k-1}) - d_k^{new}$ ;
        end ;
    end ;
```

where the superscript *new* is used only to indicate where old values are overwritten.

To see that this iteration computes the same value of u_m as (2.1) define $\delta u_k(j)$ to be the value for variable δu_k computed at outer iteration j. Assume inductively that $\Sigma_{j=0}^{k-1}(\delta u_{k-1}(j)) = u_{k-1}$ and $\Sigma_{j=0}^{k-1}(v_{k-1}(j)) = v_{k-1} + r_{k-2}$. By a sequence of variable substitutions one sees that

$$v_k^{new} := (I - A_k Rel_k)(v_k + d_{k-1} - A_k(\delta u_{k-1})).$$

If we then sum the values of $v_k(j)$ we have

$$\begin{aligned}\Sigma_{j=0}^{k}(v_k(j)) &= v_k + (I - A_k Rel_k)(\Sigma_{j=1}^{k} v_{k-1}(j-1) \\ &\qquad + \Sigma_{j=1}^{k}(d_{k-2}(j-1) - A_k(\delta u_{k-2}(j-1))) \\ &= v_k + (I - A_k Rel_k)(v_{k-1} + f_{k-1} + (A_{k-2} - A_{k-1})u_{k-1}) \\ &= v_k + r_{k-1}.\end{aligned}$$

The sum of the values

$$\begin{aligned}\Sigma_{j=0}^{k}(\delta u_k(j)) &= Rel_k(\Sigma_{j=0}^{k} v_k(j) + \\ &\qquad \Sigma_{j=0}^{k}(d_{k-1}(j) - A_k(\delta u_{k-1}(j))) \\ &= Rel_k(v_k + r_{k-1} + (A_{k-1} - A_k)u_{k-1}) = u_k.\end{aligned}$$

This decomposition of the residual and solution vector into a set of "differences" motivates the following algorithm:

```
$p_m := f_m$ ; for $k$ = 0 to m-1 do $p_k := 0$ ;
$U := 0$ ; for $k$ = 0 to m do $d_k := u_k := w_k := 0$ ;

Repeat
```

```
    for $k = m$ downto 1 do begin
        $p_{k-1} := p_{k-1} + proj(p_k)$ ;
          $v_k := p_k - proj(p_k)$ ;
        end
    $w_m := u_m$ ;

    For $k = m$ downto 1 do begin
        $u_k^{new} := w_k + u_{k-1} + Rel_k(v_k + d_{k-1} - A_k u_{k-1})$
        ;
        $d_k^{new} := A_k u_k$ ;
        $p_k^{new} := v_k + d_{k-1}^{new} + A_k w_k$ ;
        end ;
until $||f_m - A_m u_m|| <$ tolerance ;
```

To see that this is, in fact, a consistent algorithm, we proceed as follows. An examination of the first loop reveals that we have the invariant

$$\Sigma_k v_k = \Sigma_k p_k,$$

and for $k = m$,

$$p_m^{new} + d_m^{new} = v_m + d_{m-1} + d_m.$$

Summing from 1 to m

$$\Sigma_k p_k^{new} + \Sigma_k d_k^{new} = \Sigma_k v_k + \Sigma_k d_k.$$

Combining this with the first invariant and the initial conditions we have

$$\Sigma_k p_k^{new} + \Sigma_k d_k^{new} = f_m.$$

At the end of each outer iteration the "current" approximate solution is given by $\Sigma_k u_k^{new}$. It is not hard to show that, at each stage, the residual error is dominated according to the relation

$$||f_m - A_m(\Sigma_k u_k)|| < ||\Sigma_k p_k|| + ||\Sigma_k(d_{k-1} - A_k u_{k-1})||.$$

In Section 3 the term on the right is shown to be reduced at each outer iteration. In practice we never actually form the sum $\Sigma_k u_k^{new}$, but rather, we wait while u_k, $k < m$ near zero and the residual error $f_m - A_m u_m$ on the finest grid is within acceptable limits.

To see where the parallelism in the new algorithm is we consider the large scale flow structure. Rewriting the algorithm in terms of the two multivalued functions

```
function $P(f, pin : array[1..2^k, 1..2^k]$ of real)
            returns ($v : array[1..2^k, 1..2^k]$ of real,
                $pout : array[1..2^{k-1}, 1..2^{k-1}]$ of real);
    begin
        $pout := proj(f + pin)$ ;
           $v := f + pin - inj(pout)$ ;
    end ;
```

```
function $R(vin : array[1..2^k, 1..2^k]$ of real ;
            $uin, din : array[1..2^{k-1}, 1..2^{k-1}]$ of real)
    returns ($p, uout, dout$ :
            $array[1..2^k, 1..2^k]$ of real) ;
    begin
        $uout := inj(uin) +$
                $Rel(vin + din - A(inj(uin)))$ ;
        $dout := A(uout)$ ;
            $p := vin + din - dout$ ;
    end ;
```

The new algorithm takes the form

```
$p_m := f_m$ ; for $k = 0$ to m-1 do $p_k := 0$ ;
$U := 0$ ; for $k = 0$ to m do $d_k := u_k := w_k := 0$ ;

Repeat
    for $k = m$ downto 1 do $v_k, p_{k-1} := P(v_k, p_k)$ ;
    For $k = m$ downto 1 do
        $p_k, u_k, d_k := R(v_k, u_{k-1}, d_{k-1})$ ;
    $w_m := w_m + u_m$ ;
until $||f_m - A_m w_m|| <$ tolerance ;
```

The data flow graph for this computation is shown in Figure 2.3 below.

To see the additional parallelism generated in this computation that is not part of the V cycle algorithm notice that as a data flow structure, the computation can move in a wave front from the upper left corner, sweeping to the right. Another formulation is to restructure the algorithm so that it is represented as a *Macro, Data Driven Systolic Array* illustrated in Figure 2.4 below.

In this case we "unroll" the loops and view each instance of a function invocation at a given level as an instance of a set of messages to a process which executes that function when all data arguments are available. Figure 2.4 can be viewed as a folding of the dataflow graph in Figure 2.3 (actually a quotient structure).

The node denoted with Σ is used to accumulate the partial results after each invocation at the top level. All synchronization is based on inherent dataflow synchronization inherited from the diagram in Figure 2.3. In Section 4 we considerthe lower levels of parallelism that can be obtained by exploiting the structure within each of the processes invocations.

3. NUMERICAL ANALYSIS

The algorithm constructed above has several interesting numerical properties that will be described in this section. Of particular concern is, of course, the rate at which the algorithm converges to the solution, or, at least, the rate at which the residual error is reduced to zero. In order to prove anything about the algorithm we must make certain assumptions about the structure of the family of approximating operators:

$$(A_k : M_k \rightarrow R^k, k = 0 \ldots m = log(n)).$$

In particular, we will need to make two assumptions about how well A_k approximates A_{k+1}. These assumptions take the form:

rectangles. In this paper we shall not consider the problem of justifying these two properties for more complex (and interesting) grid structures such as those involving local refinement or re-entrant corners.

The basic iteration step in the algorithm takes the form shown in Figure 3.1 below. The computations can be summarized as follws: Rel_k is the approximate inverse to

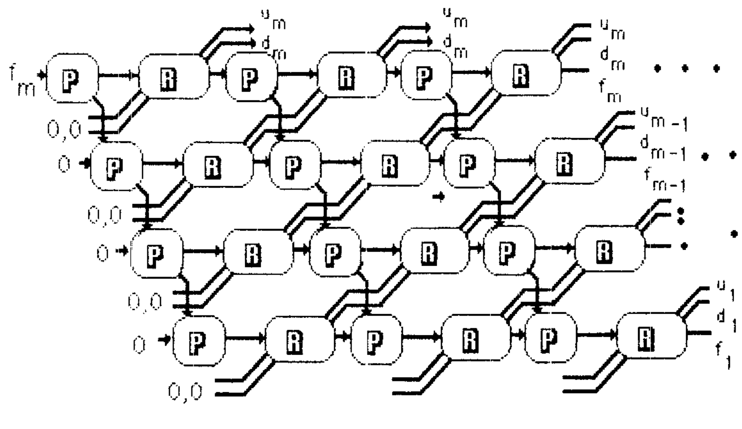


Figure 2.3: Macro Data Flow Graph

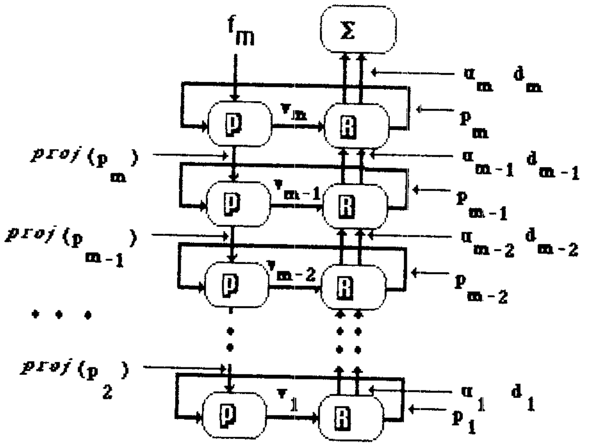


Figure 2.4: Macro Data Driven Systolic Array

Property P1: Let v be in M_k. There exists $c << 1$ such that:

$$||(A_{k+1} - A_k)v|| \le c||A_k v||.$$

Property P2: Let v be in M_{k-1}. In this case:

$$||(A_{k+1} - A_k)v|| \le 1/4||(A_k - A_{k-1})v||.$$

Property P2 states that most of the error in approximating A_{k+1} by A_{k-1} for vectors in M_{k-1} is accounted for by the failure of A_{k-1} to approximate A_k. It is tedious but not difficult to verify these claims for simple uniform grids on

A_k defined on M_k by Jacobi smoothing and P is the projection splitting operator defined in the previous section. We then have:

$$u_k^{new} = u_{k-1} + Rel_{k-1}(v_{k-1} + d_{k-1} - A_{k-1}u_{k-1})$$

$$d_k^{new} = A_{k-1}u_k^{new}$$

$$r_k = v_{k-1} + d_{k-1} - d_k^{new}$$

$$v_*^{new} = P(r^*)$$

Two important identies are derived from the definition of P. Namely,

$$\Sigma_k v_k^{new} = \Sigma_k r_k$$

and

$$f = \Sigma_k(v_k + d_k) = \Sigma_k(v_k^{new} + d_k^{new})$$

In addition to the two assumptions made above we will need one further property of the smoothing iterations that make u the relaxation operator Rel_k. In particular, all multigrid iterations are based on the idea that for vectors x in M_k that are orthogonal to M_{k-1}, the operator Rel_k is a good approximation to the inverse of A_k. We write this as:

Property 3: Let s be the number of inner iterations used in Rel_k. There exists a constant $r < 1$ such that for all x in M_k that are orthogonal to M_{k-1},

$$||(I - A_k Rel_k)s|| < r^s||x||.$$

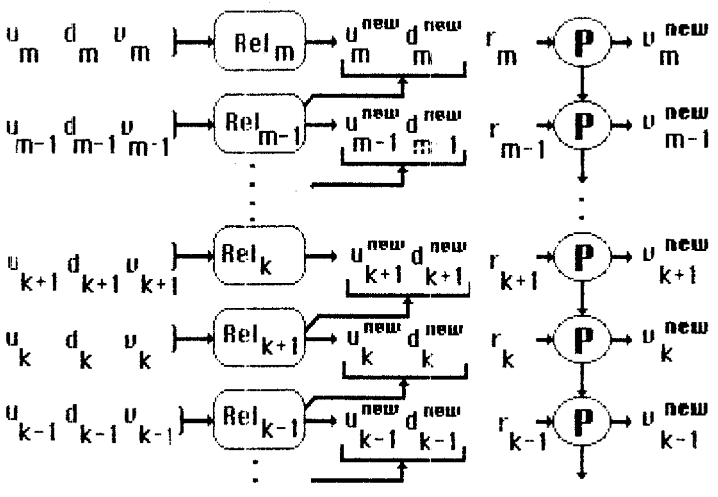


Figure 3.1: Single Stage of Parellel MG Iteration

The projection splitting operator P turns the vector family r_* into a family v_* such that for all K, v_k is in M_k and is orthogonal to the subspace M_{k-1}. In other words, v_k is orthogonal to v_j if $j <> k$ and we have $||\Sigma_k v_k||^2 = \Sigma_k ||v_k||^2$. Similarly, observe that the identity $PA_{k+1}i = A_k$ implies that for any vector x in M_k, the vector $(A_{k+1} - A_k)x$ in M_{k+1} is orthogonal to M_k. This implies:

Property P4: Let x_k be in M_k for $k = 0 \,..\, m$, then the following identity holds:

$$||\Sigma_k (A_{k+1})x_k||^2 = \Sigma_k ||(A_{k+1} - A_k)x_k||^2.$$

In order to study the rate of convergence of this algorithm we must show that there is some function of residual data after each iteration is less than the corresponding value before hand. The most natural choice for a residual function to try to minimize is $||f - A_m(\Sigma_k u_k)||$. While this has a certain esthetic appeal it is difficult to approximate because the algorithm never computes the term $\Sigma_k u_k$. On the other hand, the residual expression $||f - \Sigma_k A_k u_k||$ can be bounded by quantities that are related to some aspect of the computation:

$$\begin{aligned} ||f - \Sigma_k A_k u_k|| &= ||\Sigma(v_k + d_k - A_k u_k)|| \\ &\le ||\Sigma v_k|| + ||\Sigma(d_k - A_k u_k)||. \end{aligned}$$

We shall show that the term on the right hand side of this inequality is reduced at each iteration. More specifically,

Theorem 3.1: There exists constant $C < 1$ which depends on a value s_0 that grows as $log(m) = loglog(n)$, such that if $s > s_0$ inner iterations are done at each smoothing step. Then:

$$\begin{aligned} &||\Sigma v_k^{new}|| + ||\Sigma(d_k^{new} - A_k u_k^{new})|| \\ &< C(\Sigma||v_k|| + \Sigma||d_k - A_k u_k||). \end{aligned}$$

Proof: We first consider the second term on the left hand side. From the definition of d_k^{new} and u_k^{new} we have

$$\begin{aligned} ||\Sigma(d_k^{new} - A_k u_k^{new})|| &\le ||\Sigma(A_k - A_{k-1})u_{k-1}|| \\ &+ ||\Sigma(A_{k-1} - A_k)Rel_{k-1}q_{k-1}||, \end{aligned}$$

where we define,

$$q_k = v_k + d_k - A_k u_k.$$

From Properties 3 and 2 we have

$$\begin{aligned} ||\Sigma(A_k - A_{k-1})u_{k-1}|| &= (\Sigma||(A_k - A_{k-1})u_{k-1}||^2)^{1/2} \\ &\le 1/4(\Sigma||(A_{k-1} - A_{k-2})u_{k-1}||^2)^{1/2} \\ &\le 1/4(||\Sigma(A_{k-1} - A_{k-2})u_{k-1}||) \\ &\le 1/4(||\Sigma(d_{k-1} - A_{k-1}u_{k-1})||) \end{aligned}$$

From Properties 4 and 1 we have

$$\begin{aligned} &||\Sigma(A_k - A_{k-1})Rel_{k-1}q_{k-1}|| = \\ &\quad (\Sigma||(A_k - A_{k-1})Rel_{k-1}q_{k-1}||^2)^{1/2} \\ &\quad \le c(\Sigma||A_{k-1}Rel_{k-1}q_{k-1}||^2)^{1/2}. \end{aligned}$$

But Rel_k is an approximate inverse of A_k and we can easily show from Property 3 that $||A_{k-1}Rel_{k-1}q_{k-1}|| < 2||q_{k-1}||$. Furthermore each of the q_k are orthogonal to each other, so we have

$$\begin{aligned}||\Sigma(A_k - A_{k-1})Rel_{k-1}q_{k-1}|| &\le 2c(||\Sigma q_k||) \\ &\le 2c\,||\Sigma v_k|| + c\,||\Sigma(d_k - A_k u_k)||.\end{aligned}$$

Combining these results we have

$$||\Sigma(d_k^{new} - A_k u_k^{new})|| \le 2c\,||\Sigma v_k|| +$$

$$(2c + 14)||\Sigma(d_k - A_k u_k)||.$$

The remaining term is

$$\begin{aligned}||\Sigma v_k^{new}|| = ||\Sigma r_k|| &= ||\Sigma(v_{k-1} + d_{k-1} - A_k u_k^{new})|| \\ &= ||\Sigma(I - A_{k-1}Rel_{k-1})(v_{k-1} + d_{k-1} - A_{k-1}u_{k-1})|| \\ &\le \Sigma||(I - A_{k-1}Rel_{k-1})(v_{k-1} + d_{k-1} - A_{k-1}u_{k-1})|| \\ &\le r^s\Sigma||v_{k-1} + d_{k-1} - A_{k-1}u_{k-1}|| \\ &\le m^{1/2}r^s(\Sigma||v_{k-1} + d_{k-1} - A_{k-1}u_{k-1}||^2)^{1/2} \\ &\le m^{1/2}r^s||\Sigma(v_{k-1} + d_{k-1} - A_{k-1}u_{k-1})|| \\ &\le m^{1/2}r^s(||\Sigma v_{k-1}|| + ||\Sigma(d_{k-1} - A_{k-1}u_{k-1})||.\end{aligned}$$

Adding these two inequalities the result follows when we set $C = 1/4 + 2c + m^{1/2}r^s$ and choose s_0 so that $C < 1$.

4. ARCHITECTURAL IMPLICATIONS

There are three distinct levels of parallelism with the algorithm derived here. At this level, the algorithm was given in Section 2, we see that it is composed of linear operations on grids of size 2^k by $2^k, k = 1 \,..\, m$. In each case these operators, namely A_k, *inj*( ), and *proj*( ), consist of sparse matrix multiplications. For example if $A_k = a_k(i,j,s,t)$, then the usual definitions for piecewise bilinear finite elements has $a_k(i,j,s,t) <> 0$ if and only if $|i - s| \le 1$ and $|j - t| \le 1$. The operator A_k for a "five point difference" is given by:

```
function $A_k(u$ : $array[1 \,..\, 2^k, 1 \,..\, 2^k]$ of real)
              (returns $w$ : $array[1 \,..\, 2^k, 1 \,..\, 2^k]$
              of real) ;
    begin
        For $i, j = 1 \,..\, 2^k$ do
            $w(i,j) := a_k(i,j,i,j)u(i,j)$
             $+ a_k(i,j,i-1,j)u(i-1,j) +$
                $a_k(i,j,i,j-1)u(i,j-1)$
                 $+ a_k(i,j,i+1)u(i+1,j)$
                 $+ a_k(i,j,i,j+1)u(i,j+1)$ ;
        end ;
```

Distributing the loop makes this a linear combination of vector operations. That is let $a_k^{s,t}(i,j) = a_k(i,j,i+s,j+t)$ for $s, t = -1 \,..\, 1$, then the product is given by the vector expression:

$$\begin{aligned}A_{(u)} = a_k^{o,o}(*,*)*u(*,*) + a_k^{o,1}(*,*)*u(*,*+1) + \\ a_k^{1,o}(*,*)*u(* + 1,*) + a_k^{o,-1}(*,*)*u(*,*-1) + \\ a_k^{-1,o}(*,*)*u(*-1,*).\end{aligned}$$

Thus, at the lowest level the computation can easily exploit vector computation, but the basic operators of the algorithm involve sums such as above of five or more vector computations. From the point of view of hardware, we see that it would be easy to use processors with a large number of independently scheduled but "chainable" vector pipelines. An alternatiave would be to have a system where a processor controls as "Illiac IV" style array unit that carry out the sparse matrix operation as illustrated in Figure 4.1.

The lowest level of easily exploited paralelism is then vector operations. (This is not surprising for a scientific computation). At the next level up we see that we can do many vector operations concurrently to get the fastest parallel computation of the basic linear operators A_k ,*inj*( ) and *proj*( ). At the next level up we see that the operators P and R are easily described as simple acyclic graphs whose nodes are the array operators described above.

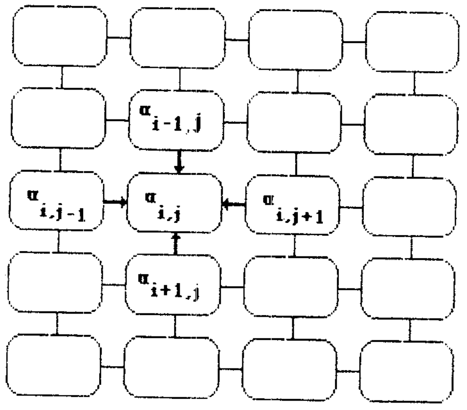
On the other hand, there is another important form of second level parallelism that is important. In many applications the multigrid structure is determined by a local refinement process. One approach to building grid structures that are easily fitted to complex geometries and are easily refined is to use a simple uniform building block such as a 2^k by 2^k grid for some fixed value of k. As illustrated in Figure 4.2 a wide variety of structures can be generated from these objects.

If we define the basic array operation to be a matrix product of a sparse array times a solution or data vector defined on one of the uniform subgrids, then the higher operators like P and R can be expressed as a computation on a network of connected grids defined at the same level in the refinement graph. This idea is discussed in greater detail in [GaVR83].

At the highest level, as we have already seen in Section 2, the computation can be seen as a "large grain" macro systolic array. As with the lower levels the synchronization is all based on dataflow concepts, but at the highest level, the unit of data is a complete array data associated with one complete grid level. The advantage of this is twofold. First, the large granularity allows us to schedule rather large computations on processors at one time and any associated overhead will appear small. The second advantage is that if each processor has the ability to exploit lower levels of parallelism, (as illustrated in the diagram in Figure 4.3 below) then a "local controller" can schedule the fine grain computation. We have, therefore, substructured the scheduling problem.

5. REFERENCES

[Bran80] Brandt, A., "Multigrid Solvers on Parallel Computers", ICASE NASA Langley Research Center, Hampton, Virginia Report No. 80-23, 1980.



$$A(\alpha_{i,j}) = a_{i,j}u_{i,j} + a_{i-1,j}u_{i-1,j} + a_{i+1,j}u_{i+1,j} + a_{i,j-1}u_{i,j-1} + a_{i,j+1}u_{i,j+1}$$

Figure 4.1: SIMD Array Multiplication Structure

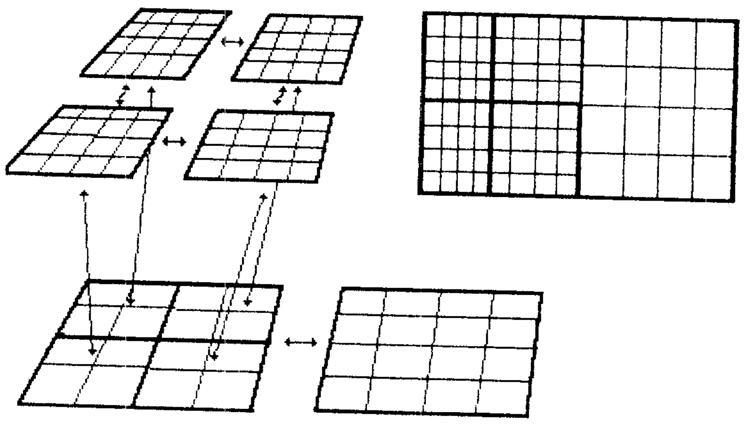


Figure 4.2: Locally Refined, Block Structured Grids

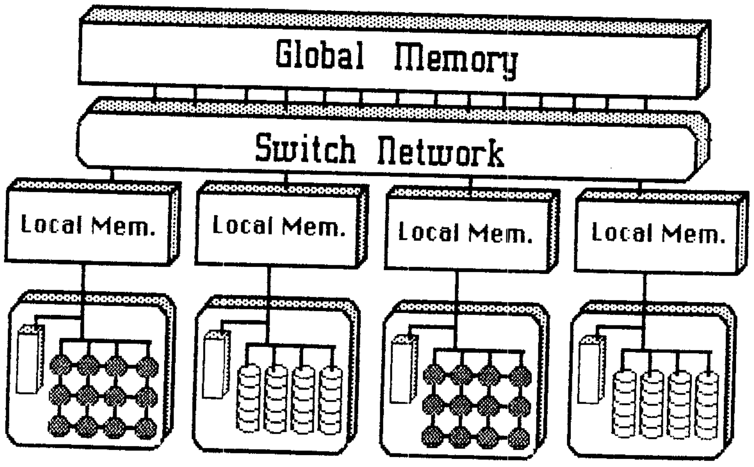


Figure 4.3: Cluster of Low Level Parallel Structures

[GKLS83] Gajski, D., Kuck, D., Lawrie, D., Sameh, A., (1983) "CEDAR, A Large Scale Multiprocessor," Technical Report, Cedar project, Dept. of Comp. Science, University of Illinois at Urbana-Champaign.

[GaVR83] Gannon, D., Van Rosendale, J. (1983), "Highly Parallel Multigrid Solvers for Elliptic PDEs: An Experimental Analysis," ICASE Report 82-36, ICASE, NASA Langley Research Center, Hampton, Virginia.

[GaVR84] Gannon, D., and Van Rosendale, J., "Parallel Architectures for Iterative Methods on Adaptive, Block Structured Grids," in **Elliptic Problem Solves II**, Birkoff and Schoenstadt eds. Academic Press (1984) pp. 93-104.

[Kung80] Kung, H.T., Leiserson, C.E. (1980), "Algorithms for VLSI Processor Arrays," in Mead and Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, Massachusetts, pp. 271-292.

[Nico77] Nicolaides, R. A., "On the L2 Convergence of an Algorithm for Solving Finite Element Equations", Math. Comp. 31, 1977, 892-906.

[VRos80] Van Rosendale, J. R., "Rapid Solution of Finite Element Equations on Locally Refined Grids by Multi-Level Methods", Department of Computer Science, University of Illinois, UIUCDCS-R-80-1021, Urbana, Illinois, 1980.