Kragujevac J. Math. 25 (2003) 5–18.

PARALLEL METHODS FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Ioana Chiorean

Babeş-Bolyai University, Department of Mathematics, Cluj-Napoca, Romania

(Received May 28, 2003)

Abstract. The aim of this paper is to study some iterative methods for solving Partial Differential Equation, like Jacobi, Gauss-Seidel, SOR and multigrid, making a comparison among them from their computational complexity point of view.

1. INTRODUCTION

Various problems arising from Physics, Fluid Dynamics, Chemistry, Biology, etc. can be modeled mathematically by means of partial differential equations. It is known that, sometimes, the exact solution (or solutions) is difficult to be determined, so one has to compute an approximation of it, generated by means of the approximate problem attached to the continuous one.

In order to solve the approximate problem, obtained by discretizing the initial, continuous problem, several numerical methods can be used: direct (e.g. Gaussian elimination, factorization techniques, etc.) or iterative (e.g. Jacobi, Gauss-Seidel, SOR, multigrid, etc.).

Sometimes the direct solvers are preferred, but if the problem is too large, the iterative ones are more appropriate. It seems to be more attractive, too, from the computation point of view, if more than one processor are used, it means from the parallel calculus point of view (see [6], [7]).

The aim of this paper is to make a review of some parallel abordations of the Jacobi, Gauss-Seidel, SOR and multigrid methods, emphasizing the last one and trying to reduce its computational complexity by using a different type of communication among processors. In order to present these ideas, the Poisson's equation will be used as the model problem.

2. REVIEW OF THE DISCRETE POISSON'S EQUATION

This equation may arise in heat flow, electrostatics, gravity, etc. and, in 2-dimensions, is:

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y) \text{ in } \Omega \tag{1}$$

with Ω, let's say, the unit square $0 < x, y < 1$, and with some boundary conditions, let's consider the simplest case

$$u(x,y) = 0 \text{ on } \partial\Omega \tag{2}$$

This is the continuous problem we have to solve. We discretize this equation by means, e.g., of finite differences (see [3]). We use an $(n+1) \times (n+1)$ grid on Ω (it means on the unit square), where $h = \dfrac{1}{n+1}$ is the grid spacing. Let's denote u_{ij} the approximate solution at $x = ih$ and $y = ih$. This is shown in fig.1, for $n = 7$.

Denoting $b_{ij} = -f(ih, jh) \cdot h^2$, the approximate problem becomes:

$$4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = b_{ij} \tag{3}$$

for all $1 \leq i, j \leq n$.

In the matrix form, (3) is a linear system of $N = n^2$ equations with N unknowns, let's write it

$$A \cdot u = b. \tag{4}$$

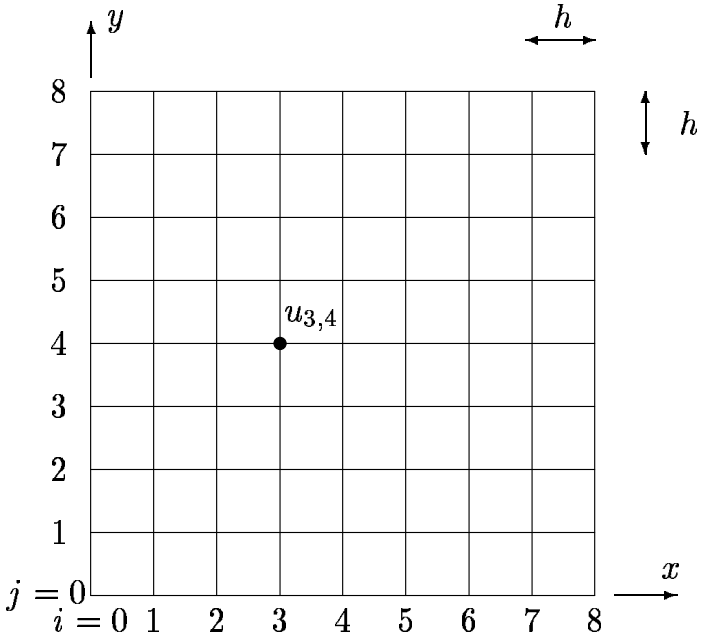


Fig.1. The discretized domain $\overline{\Omega}$ with $n = 7$

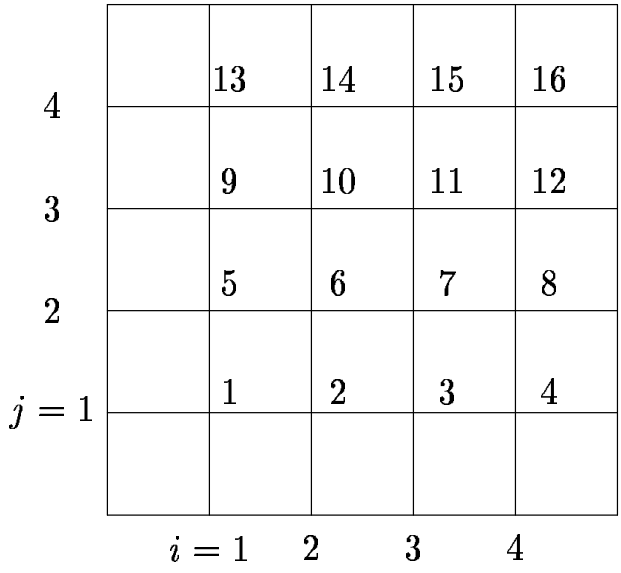


Fig.2. Linearized order of unknowns on a 2D grid

Using the linearized order of unknowns on a 2D grid, as in fig.2, the system will be:

$$\left[\begin{array}{cccc|cccc|cccc|cccc}
4 & -1 & & & -1 & & & & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & & & & \\
 & -1 & 4 & -1 & & & -1 & & & & & & & & & \\
 & & -1 & 4 & & & & -1 & & & & & & & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & & & & & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & & & & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & & & & & \\
 & & & -1 & & & -1 & 4 & & & & -1 & & & & \\
\hline
 & & & & -1 & & & & 4 & -1 & & & -1 & & & \\
 & & & & & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & & & & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & & & & & -1 & & & -1 & 4 & & & & -1 \\
\hline
 & & & & & & & & -1 & & & & 4 & -1 & & \\
 & & & & & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & & & & & -1 & & & -1 & 4 & -1 \\
 & & & & & & & & & & & -1 & & & -1 & 4
\end{array}\right]
*
\left[\begin{array}{c}
u_{11} \\ u_{21} \\ u_{31} \\ u_{41} \\ \hline
u_{12} \\ u_{22} \\ u_{32} \\ u_{42} \\ \hline
u_{13} \\ u_{23} \\ u_{33} \\ u_{43} \\ \hline
u_{14} \\ u_{24} \\ u_{34} \\ u_{44}
\end{array}\right]
=
\left[\begin{array}{c}
b_{11} \\ b_{21} \\ b_{31} \\ b_{41} \\ \hline
b_{12} \\ b_{22} \\ b_{32} \\ b_{42} \\ \hline
b_{13} \\ b_{23} \\ b_{33} \\ b_{43} \\ \hline
b_{14} \\ b_{24} \\ b_{34} \\ b_{44}
\end{array}\right]$$

3. CLASSICAL RELAXATION SCHEMES

We include here Jacobi, Gauss-Seidel and SOR methods.

Denoting by k and $k+1$ two successive iterations, these methods derive from (3) in the following way:

$$u_{ij}^{(k+1)} = (u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + b_{ij})/4 \tag{5}$$

$$u_{ij}^{(k+1)} = (u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + b_{ij})/4 \tag{6}$$

$$u_{ij}^{(k+1)} = u_{ij}^{(k)} + \omega(u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + b_{ij} - 4u_{ij}^{(k)})/4 \tag{7}$$

(with $1 < \omega < 2$ to have over-relaxation).

In [4], parallel alternatives for these methods are given, when a lattice connectivity among processors is used. A comparison among their computational complexities is made, which can be sumarized in the following table:

p = number of processors

f = time per flop

α = startup for a message

β = time per word in a message

$$\begin{aligned} \text{Complexity (Jacobi)} &= \text{number of steps} * \text{cost per step} = \\ &= O(N) * ((N/p)f + \alpha + (n/p)\beta) = \\ &= O(N^2/p)f + O(N)\alpha + O(N^{\frac{3}{2}}/p)\beta \end{aligned}$$

$$\begin{aligned} \text{Complexity (SOR)} &= \text{number of steps} * \text{cost per step} = \\ &= O(\sqrt{N}) + ((N/p)f + \alpha + (n/p)\beta) = \\ &= O(N^{\frac{3}{2}}/p)f + O(\sqrt{N})\alpha + O(N/p)\beta \end{aligned}$$

Remark. The Gauss-Seidel method converges twice as fast as Jacobi, but requires twice as many parallel steps, using the "checkerboard" ordering of nodes (see [4], [2], [3]), so about the same run time, in practice. This is the reason why it does not appear in the table.

In [3] we show that changing the connectivity among processors, and using a ring communication, the cost per step for SOR method can be made lower, and then the whole complexity of the SOR parallel method.

4. THE MULTIGRID METHODS

It is known (see e.g. [1], [5]) that multigrid is a divide-and-conquer algorithm for solving a discrete problems. It is widely used on partial differential equations, as well. It is divide-and-conquer in two related senses. First, it obtains an initial solution for an $(n \times n)$ grid using an $\left(\frac{n}{2} \times \frac{n}{2}\right)$ grid as an approximation, taking every other grid point from the $(n \times n)$ grid. The coarser $\left(\frac{n}{2} \times \frac{n}{2}\right)$ grid is in turn approximated by an $\left(\frac{n}{4} \times \frac{n}{4}\right)$ grid, and so on recursively.

The second way multigrid uses divide-and-conquer is in the frequency domain. This requires us to think of the error as a sum of sine-curves of different frequencies.

Then the work we do on a particular grid will eliminate the error in half of the frequency components not eliminated on other grids.

Without loss of generality, one consider a $(2^m - 1) \times (2^m - 1)$ grid of unknowns and adding the nodes at the boundary, which have the given value 0, one get a $(2^m + 1) \times (2^m + 1)$ grid on which the algorithm will operate. Let's denote $n = 2^m + 1$. Also, let $P(i)$ denote the problem of solving Poisson's equation on a $(2^i + 1) \times (2^i + 1)$ grid, with $(2^i - 1) \times (2^i - 1)$ unknowns. The problem is specified by the grid size i, the coefficient matrix A_i and the right hand side, b_i. A sequence of related problems $P(m), P(m-1), \ldots, P(1)$ on coarser and coarser grids are generated, where the solution to $P(i-1)$ is a good approximation to the solution of $P(i)$. Some grids for $n = 9$ are shown in fig.3.

If we denote b_i the right hand side of the linear system $P(i)$ and x_i an approximate solution of $P(i)$ (thus x_i and b_i are $(2^i - 1) \times (2^i - 1)$ arrays of values at each grid point), the basic *Multigrid V-cycle* is (MGV):

function $MGV(b_i, x_i)$ {return an improved solution} x_i to $P(i)$

if $i = 1$ {only one unknown}

compute the exact solution x_1 on $P(1)$

return (b_1, x_1)

else

$x_i = S_i(b_i, x_i)$ {improve the solution}

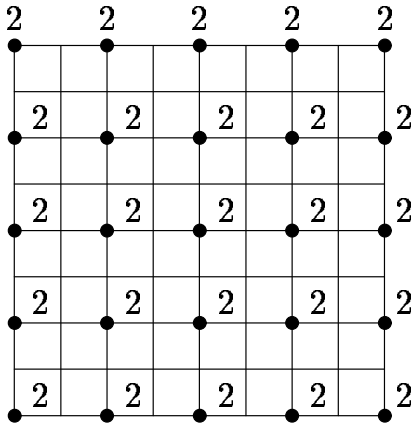
$(b_i, d_i) = I_{i-1}(MGV(R_i(b_i, x_i)))$ {solve recursively}

$x_i = x_i - d_i$ {correct fine grid solution}

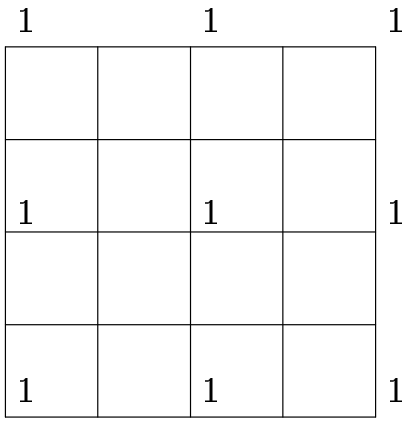
$x_i = S_i(b_i, x_i)$ {improve solution some more}

return (b_i, x_i)

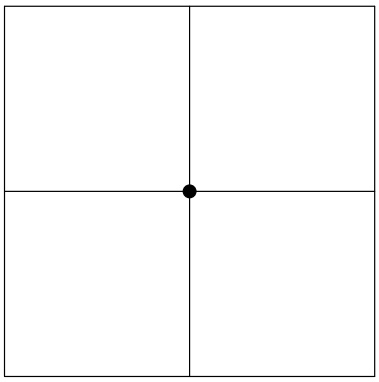
endif



P(3): 9×9 grid of points
7×7 grid of unknowns
Points labeled 2 are part
of next coarser grid



P(2): 5×5 grid of points
3×3 grid of unknowns
Points labeled 1 are part
of next coarser grid



P(1): 3×3 grid of points
1×1 grid of unknowns

Fig.3. Fine and coarse grids for $n = 9$

Remark 1. S_i denotes the smoothing operator (see e.g. [1]), which finally is one or more relaxation steps; I_{i-1} is the prolongation operator which takes an approximate solution x_{i-1} for $P(i-1)$ and converts it to an approximation x_i for the problem $P(i)$

on the next finer grid; R_i is the restriction operator, which maps the approximate solution for $P(i)$ to the next coarser grid; d_i is the defect, it means how much the solution x_i fails in verifying the system.

Remark 2. The algorithm is called V-cycle because if we draw it schematically in space and time (with grid number i), with a point for each recursive call to MGV, it lookd like fig.4, starting with a call to $MGV(P(5), x_5)$ in the upper left corner. This calls MGV on grid 4, then 3, and so down to the coarsest grid 1, and then back up to grid 5, again.

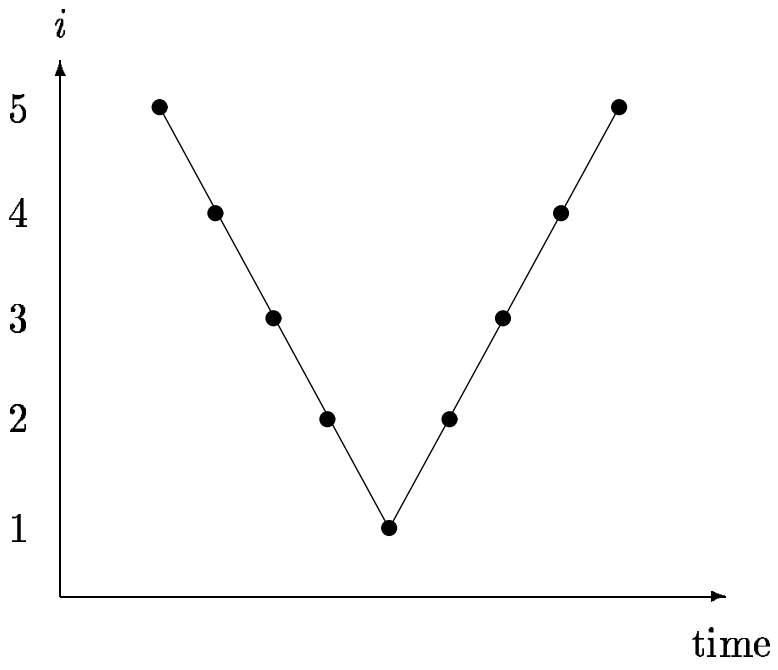


Fig.4. Multigrid V-Cycle on 5 grids

If we perform the algorithm on a serial computer (so with one processor), the complexity of MGV can be determined, in the terms of big-Oh, in the following way: we observe that the work at each point in the algorithm is proportional to the number of unknowns, since the value at each grid point is just averaged with its nearest neighbors. Thus, each point at grid level i on the "V" in the V-cycle will cost

$(2^i - 1)^2$, which is of order $O(4^i)$ operations. If the finest grid is at level m, the total serial work will be given by the geometric sum

$$\sum_{i=1}^{m}(2^i - 1)^2, \text{ which is of order } O(4^m)$$

so the total serial work is proportional to the number of unknowns. In general, is of order $O(N)$, with $N = n^2$.

Remark. In [4], the Full Multigrid algorithm which uses Multigrid V-cycle as a building block is also studied. We do not insist here on it, because it is shown that it has the same serial complexity like the Multigrid V-cycle.

5. THE COMPLEXITY OF PARALLEL MULTIGRID METHOD

We know that multigrid requires each grid point to be updated depending on as many as 8 neighbors (those to the N, E, W, S, NW, SW, SE and NE). [4] studies the case in which a lattice of processors is used in order to execute the multigrid algorithm. So, having a $n = (2^m + 1) \times (2^m + 1)$ grid of data, one suppose that this is laid out on an $s \times s$ grid of processors (so $p = s^2$ processors), with each processor owning an $\left(\frac{n-1}{s} \times \frac{n-1}{s}\right)$ subgrid. This situation is illustrated in the fig.5, taking into account a 33×33 mesh, with 4×4 processor grid.

The grid points in the top processor row have been labeled by the grid number i of the problem $P(i)$ in which they participate. There is exactly one point labeled 2 per processor. The only grid point in $P(1)$ with a nonboundry value is owned by the processor above the coloured one. In the lower half of the mesh, grid points labeled m need to be communicated to the coloured processor in problem $P(m)$ of multigrid. The coloured processor owns grid points inside the coloured box, and will communicate with his neighbours in the following way: to update its own grid points for $P(5)$, it requires 8 grid point values from its N, D, E and W neighbors, as well as single point values from its NW, SW, SE and NE neighbors. Similarly, updating the values for $P(4)$, it requires 4 grid point values from the N, S, E and W neighbors, and one each from the NW, SW, SE and NE neighbors. This pattern continues until

each processor has only one grid point. After this, only some processors participate in the computation, requiring one value each from 8 other processors.

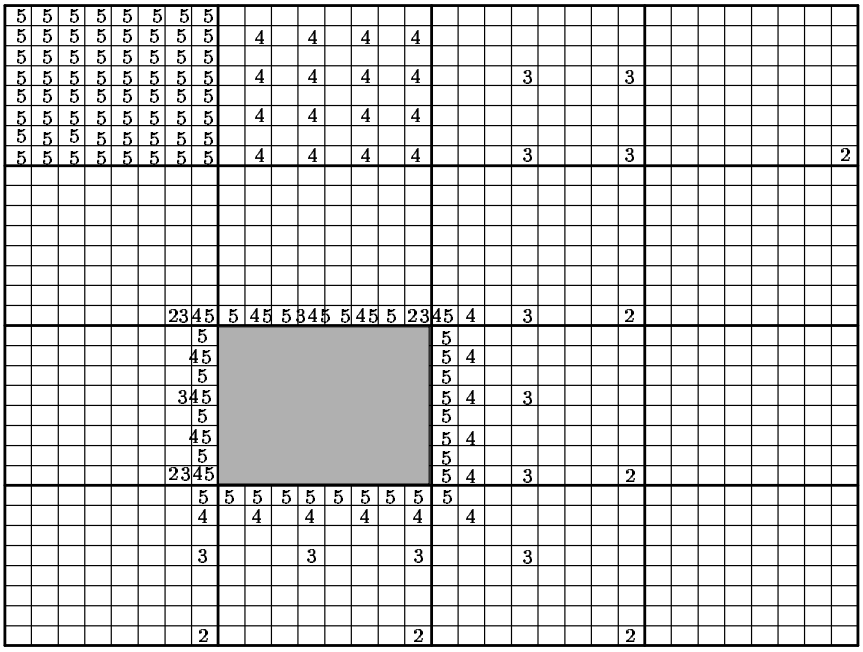


Fig.5. Multigrid on 33×33 mesh with 4×4 processor grid

Making a big-Oh analysis of the computation costs, for simplicity, let's consider $p = 4^k = 2^{2k}$, so each processor owns a $(2^{m-k} \times 2^{m-k})$ subgrid. Consider a V-cycle starting at level m. Denoting with f – the time per flop, α – the time for a message startup and β – the time per word to send a message, the following study of complexity can be made: (see [4]).

At the levels k to m

• Time at level i is:

$O(4^{i-k}) * f+$ (number of flops, proportional to the number of grid, points per processor)

$+O(1) * \alpha+$ (send a constant number of messages to neighbors)

$+O(2^{i-k}) * \beta$ (number of word sent)

Summing all these terms for $i = k$ to m, yields

• Time at levels k to m is:

$O(4^{m-k}) * f+$

$+O(m-k) * \alpha+$

$+O(2^{m-k}) * \beta =$

$= O(n^2/p) * f+$

$+O(\log(n/p)) * \alpha+$

$+O(n/\sqrt{p}) * \beta$

At level $k-1$ *to 1*

Because in levels $k-1$ through 1, fewer than all processors will own an active grid point, some processors will remain idle, and then the time complexity differs:

• Time at level i is:

$O(1)*f+$ (number of flops proportional to the number of grid points per processor)

$+O(1) * \alpha+$ (send a constant messages to neighbors)

$+O * 1) * \beta$ (number of words sent)

Summing all this for $i = 1$ to $k-1$, yields

• Time at level 1 to $k-1$ is:

$O(k-1) * f+$

$O(k-1) * \alpha+$

$O(k-1) * \beta =$

$= O(\log(p)) * f+$

$+O(\log(p)) * \alpha+$

$+O(\log(p)) * \beta$

So, the total time for a V-cycle starting at the first level is therefore

Time:

$O(n^2/p + \log(p)) * f+$

$+O(\log(n)) * \alpha+$

$+O(n/\sqrt{p} + \log(p)) * \beta$

Remark. Denoting $N = n^2$ the number of unknowns, we can state that, for $p \ll N$, the speed up of the serial multigrid is nearly perfect, but if we have enough

processors, it means at $p = N$, it reduces to $\log^2(N)$.

Some improvements in the speed up of the multigrid V-cycle method can be made if the cost of communication per step is made lower. It can be done if we use another connectivity among processor, different from the lattice one, for instance a tree network.

Let's consider, firstly, the 1D situation, with $n = 9$. In fig.6 we see the grids used in computation:

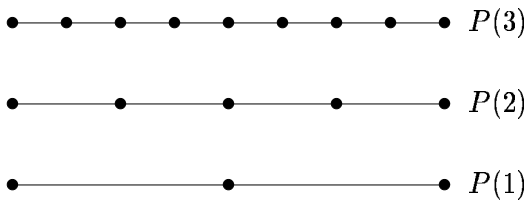


Fig.6. The grids used in computing problems $P(m)$, $m = \overline{1,3}$

We have $n = 2^m + 1$ data, (with $m = 3$ in our example) on the finest grid. Let's memorize the data in $p = n$ processors leaves of a m-ary tree, like that in fig.7.

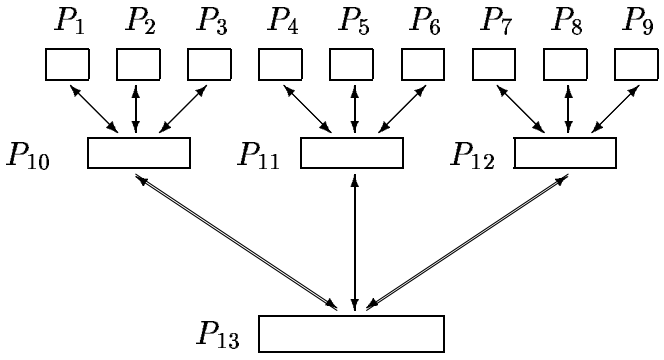


Fig.7. The tree communication network, with $m = 3$

Each processor at a level k computes one unknown, using the values of three processors from the level $k + 1$, and sending the result to processor $k - 1$.

• Time at level k is:

$O(1) * f+$ (number of flops proportional to the number of grid points per processors)

$+O(1) * \alpha+$ (send a constant number of messages to neighbor)

$+O(1) * \beta$ (number of word sent)

Summing all these terms for $k = 1$ to m, yields

• Time at level 1 to m is:

$O(m) * f+$

$+O(m) * \alpha+$

$+O(m) * \beta =$

$= O(m) * (f + \alpha + \beta) = O(\log(n)) * (f + \alpha + \beta)$

Remark. The complexity in this way is also of order $O(\log(n))$. One can show that the result is similar for the 2D case, using a more complex tree communication.

6. CONCLUSIONS

Comparing Jacobi, Gauss-Seidel, SOR and multigrid methods for solving discrete Poisson's equation on a $n \times n$ grid of $N = n^2$ unknowns for complexity point of view, we see that the best method is the multigrid. But there are also other iterative methods (of Krylov type like FFT) which generate same good times of execution.

References

[1] Briggs, W., *A multigrid tutorial*, SIAM, 1987.

[2] Chiorean, I., *On some 2D Parallel Relaxation Schemes*, Proceeding of ROGER2002, Sibiu, 2002, Burg-Verlag, 2002, 77-85.

[3] Chiorean, I., *On the complexity of some relaxation schemes*, Proc. of ICAM3, Borşa, Oct. 2002, Bul. Şt. Univ. Baia-Mare, Ser. B, Vol. XVIII (2002), nr. 2, 171-176.

[4] Demmel, J., *Lecture Notes on Numerical Linear Algebra*, Berkeley Lecture Notes in Mathematics, UC Berkeley, 1996.

[5] Hackbusch, W., *An Introduction to Multigrid Method*, Springer Verlag, 1983.

[6] Kumar, V. et al., *Introduction to Parallel Computing*, The Benjamin Cumming Pub. Company, 1994.

[7] Modi, J. J., *Parallel Algorithms and Matrix Computation*, Oxford, 1987.