Bul. Stiinţ. Univ. Baia Mare, Ser. B,
Matematică-Informatică, Vol. XVIII(2002), Nr. 2, 171 - 176

ON THE COMPLEXITY OF SOME RELATION SCHEMES

Ioana CHIOREAN

Abstract. The aim of this paper is to make a comparison among the complexity of parallel relaxation schemes (Jacobi, Gauss-Seidel, SOR), according with different way of numbering the nodes of the mesh used in computation.

MSC 2000: 35J05, 65F10, 68W10.
Keywords: PDE, Poisson equation, iterative methods for solving systems, parallel algorithms, complexity of calculus.

1. Introduction

Several problems arising from heat flow, electrostatics, gravity, etc. are modeled by means of partial differential equations. Generally, the mathematical model is complicated, that is why sometimes the exact solution is hard to obtain, analytically. An approximation of it can be computed, using a discretization method to get the approximate problem, and then solving this by means of direct or iterative methods.

Taking into account that for large dimension problems, the direct methods are not very efficient, in this paper we shall be concerned with some iterative methods, called relaxation schemes, like Jacobi, Gauss-Seidel, SOR.

We discuss the ideas using a model problem: the 2D Poisson's equation.

2. Review of the Discrete Poisson Equation

Like in [3], the 2D model problem where on the unit square Ω

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y} = f(x,y) \text{ on } \Omega; \qquad u(x,y) = 0 \text{ on } \partial\Omega \tag{1}$$

The discretized Poisson's equation is:

$$4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = b_{ij} \tag{2}$$

Equation (2) must hold for $1 \leq i, j \leq n$, and generates $N = n^2$ equations in N unknowns. When (i,j) is adjacent to a boundary ($i = 1$ or $j = 1$ or $i = 1$ or $j = 1$),

one or more of the $u_{i\pm1,j\pm1}$ values is on the boundary, and therefore equal to 0. $b_{ij} = -f(ih, jh) \cdot h^2$, the scalled value of the *rhs* function $f(x,y)$ at the corresponding grid point (i,j).

As a linear system in a more standard way, (2) can be written:

$$\hat{P} \cdot \hat{U} = \hat{b}, \tag{3}$$

where $\hat{U}$ and $\hat{b}$ are column vectors. Again according with [3] the system (3) has different writings, depending on the ordering of the nodes.

3. Jacobi's Method

It is known that Jacobi's algorithm can be derive from (2), in this way:

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

Then, at the $(k+1)$ iteration,

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

with $u_{ij}^{(0)}$ an initial approximation.

It is clear that the convergence gets slower for large problems, with the number of steps proportional to $N = n^2$ (for more details, see [3]). This make the serial complexity of Jacobi's method:

$$\begin{aligned} \text{Serial complexity} &= \text{number of steps} * \text{cost per step} = \\ &= O(N) * O(N) = \\ &= O(N^2). \end{aligned}$$

On a parallel machine, with enough processors p (it means $p \geq N$), the cost per step is $O(1)$, then the

$$\begin{aligned} \text{Parallel complexity1} &= O(N) * O(1) = \\ &= O(N). \end{aligned}$$

When $p \ll N$, we use the domain decomposition technique to distribute the values u_{ij} among processors, every processor owning a number of n^2/p grid points. Denoting, then, with p = number of processors, f = time per flop, α = startup for a message and β = time per word in a message, then

$$\begin{aligned} \text{Parallel complexity2} &= O(N) * ((N/p) * f + \alpha + (n/p)\beta) = \\ &= O(N^2/p) * f + O(N) * \alpha + O(N^{\frac{1}{2}}/p) * \beta \end{aligned}$$

where $O(N/p)$ flops are needed to update local values, α is the startup cost of messages to each neighbor and $O(n/p)$ values are communicated to neighbors. In fig.1 we have the case of $n = 7$ and $p = 4$.

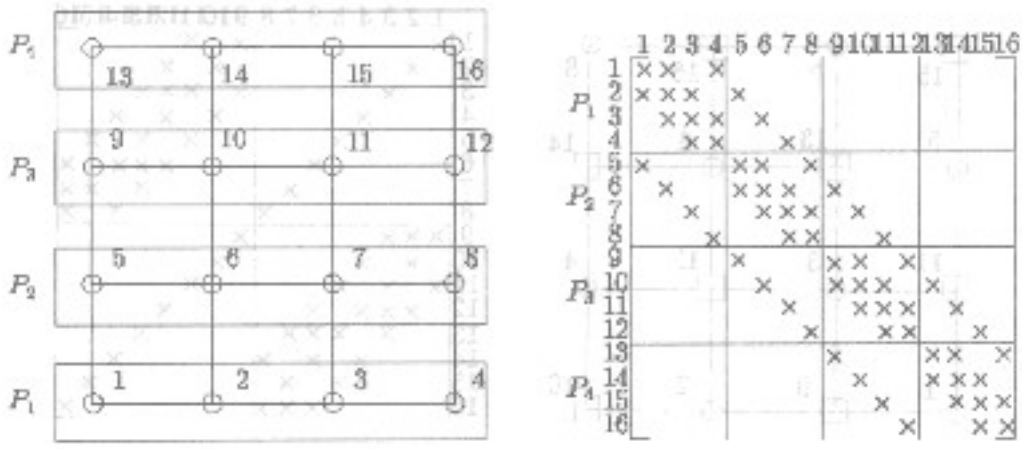

Fig.1. Dividing the domain among processors

With the above notation, processor 1 will update the lines 1, 2, 3 and 4, processor 2, the lines 5, 6, 7 and 8, and so on. The amount of communication depends only on the number of boundary nodes.

4. Gauss-Seidel Method

It is well-known that this method is similar with Jacobi, but the linear combination of the neighbors and the order of updates are different. So,

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

It is also known that this method converges twice as fast as Jacobi, so the

$$\text{Serial complexity} = \text{number of steps} * \text{cost per step} =$$
$$= O\left(\frac{N}{2}\right) * O(N) = O\left(\frac{N^2}{2}\right)$$

To parallelize (7) turns out to be not trivial, because every processor (considering $p > N$) has to wait for updated values from two other processors, which can generate a bottleneck.

But the situation can be changed if we reorder the nodes, it means the number of a line of values u_{ij} in the way indicated in fig.2.

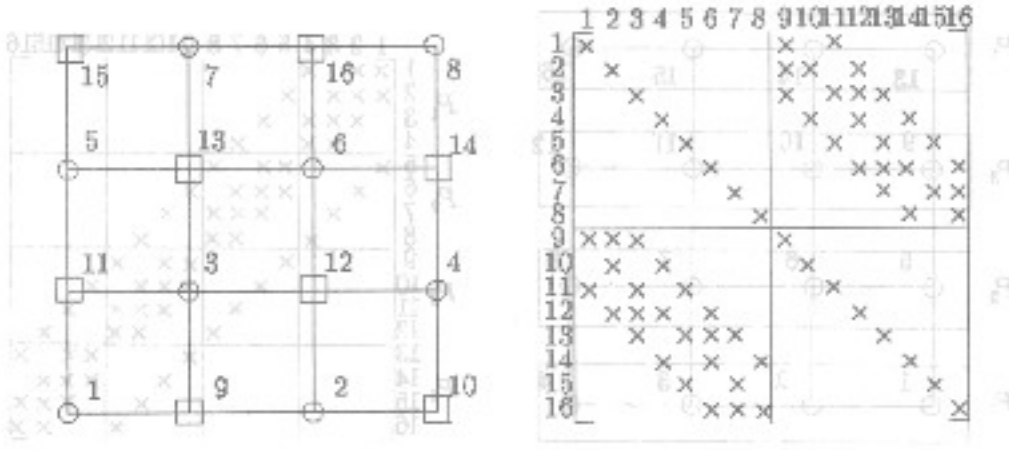

Fig.2. "Checkerboard" ordering

The idea is to update all the black points, followed by all the white points. The black grid points are connected only to white grid points, and so can all be updated simultaneously using the most recent white values. Then the white values can be updated using the most recent black values. But this idea does not accelerate convergence, only the storage of memory is improved. In order to improve the rate of convergence, other ways have to be found.

5. SOR Method

The SOR Method generates the formulas:

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

where $0 < \omega < 2$ is the relaxation factor. (For $0 < \omega < 1$ we have under-relaxation, and for $1 < \omega < 2$ we have over-relaxation).

To study the convergence of the method is made in [3].

It can be shown that the number of steps required by SOR method is approximately the square root of the number of steps Jacobi's method requires. Then, the complexity analysis shows that:

$$\begin{aligned} \text{Parallel complexity1} &= \text{number of steps} * \text{cost per step} = \\ &= O(\sqrt{N}) * ((N/p) * f + \alpha + (n/p) * \beta) = \\ &= O(N^{\frac{1}{2}}/p) * f + O(\sqrt{N}) * \alpha + O(N/p) * \beta, \end{aligned}$$

where $O(N/p)$ flops are needed to update local values, α is the startup cost of messages to each neighbor, and $O(n/p)$ boundary values are communicated to neighbors.

In analyzing the parallel complexity, the same "checkerboard" ordering was considered.

But the complexity of the parallel algorithm can be improved, using another connectivity among processors, different from that of a lattice, used since now (see [1]). This idea can be used when we do not have enough processors. If we order the nodes in the way like in fig.3.

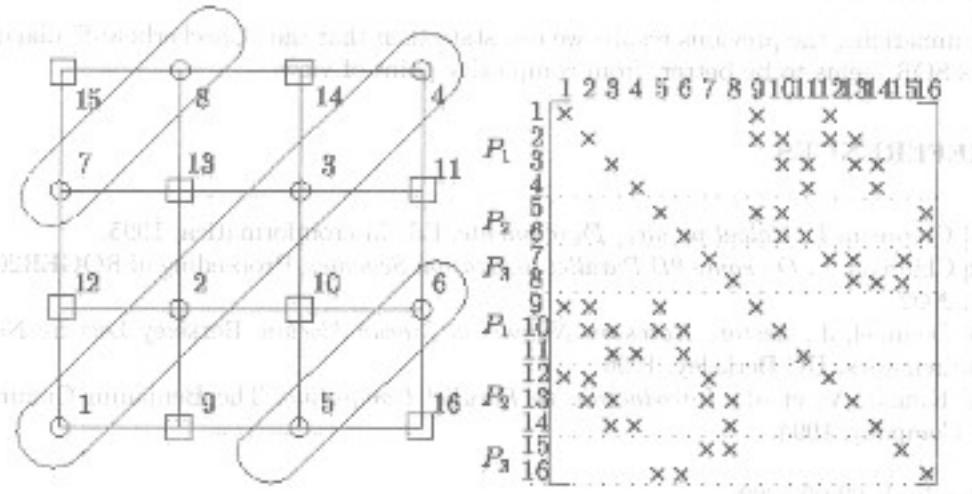

Fig.3. Red-black diagonal-blocks ordering of nodes

Let's call this ordering: "Checkerboard" diagonal blocks.

The processors will work in the following way: (in our example, we have $p = 3$ processors, connected linear, see [1]):

for $m := 1$ to p in parallel do

{compute $a_{ij}^{(k+1)}$ values for the black blocks}

end for;

for $m := 1$ to p in parallel do

{compute $a_{ij}^{(k+1)}$ values for the white blocks}

end for.

The number of messages passed among processors under this connectivity depends on the number of nodes, and this diagonal ordering reduces the number of nodes which have to be send, so the number of boundary values will be $O\left(\frac{\frac{n}{2}}{p}\right)$, it means $O(n/2p)$, because every processors has in his memory a part of the new values and has to obtained only half from the neighbors. So, the parallel complexity in this case will be:

Parallel complexity2 = number of steps * cost per step =

$$= O(\sqrt{N}) * ((N/p) * f + \alpha + (n/(2*p)) * \beta) =$$
$$= O(N^{\frac{3}{2}}/p) * f + O(\sqrt{N}) * \alpha + O(N/(2*p)) * \beta.$$

6. Conclusion

Summarizing the previous results we can state then that the "Checkerboard" diagonal blocks SOR seems to be better, from complexity point of view.

REFERENCES

[1] Chiorean, I., *Calcul paralel. Fundamente*, Ed. Microinformatica, 1995.

[2] Chiorean, I., *On some 2D Parallel Relaxation Schemes*, Proceeding of ROGER2002, Sibiu, 2002.

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

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

Received: 02.09.2002

Babes-Bolyai University Cluj-Napoca
Department of Applied Mathematics
Kogalniceanu, I, 3400 Cluj-Napoca
ROMANIA
E-mail:robica@personal.ro