International Journal of Mathematics And Its Applications

Vol.2 No.1 (2014), pp.57-65

ISSN: 2347-1557(Online)

Parallel Computing in Solving Partial Differential Equations

Mukesh Punia[†,1]

[†]Department of Mathematics, S D (PG) College, Panipat, Haryana, India

---

Abstract : The purpose of this research is to investigate many iterative techniques for solving partial differential equations, such as Jacobi, Gauss-Seidel, SOR, and multi grid, and to make a comparison among these approaches with regard to the amount of computing complexity that each of them requires.

Keywords : Partial differential equations, Gauss Jacobi Method, Jacobi Method. mappings.

---

1 Introduction

By using partial differential equations, one is able to develop mathematical models for a variety of issues that arise in fields such as physics, fluid dynamics, chemistry, biology, and so on. It is common knowledge that in some circumstances, determining the precise solution (or solutions)

[1]Corresponding author email: sapraeducation9@gmail.com (Mukesh Punia)

might be challenging. As a result, it is necessary to calculate an estimate of the solution, which is formed by the approximate problem that is tied to the continuous issue. Direct numerical methods (such as Gaussian elimination, factorization techniques, and so on) or iterative numerical methods (such as Jacobi, Gauss-Seidel, SOR, multigrid, and so on) may be used to solve the approximation issue that is created from discretizing the original continuous problem. These numerical methods can be employed in a variety of ways. If the issue is too huge, the iterative solutions are more suited, however sometimes the direct solvers are chosen. It seems to be more appealing, too, from the computing point of view, which means from the parallel calculus point of view, if more than one processor are employed (see [6], [7]). This is because the parallel calculus looks to be more efficient. The purpose of this work is to conduct a review of certain parallel abordations of the Jacobi method, the Gauss-Seidel method, the SOR technique, and the multigrid method, with an emphasis on the multigrid approach and an attempt to minimize the computational complexity of the latter by using a new style of communication across processors. The equation of Poisson will serve as the model problem for the presentation of these topics.

2 Review of the Discrete Poisson'S Equation

This equation may appear in situations involving heat flow, electrostatics, gravity, and other similar phenomena; in two dimensions, it is as follows:

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y) \quad in \quad \Omega \tag{2.1}$$

with Ω, let's say, the unit square $0 < x,\ y < 1$, And with those limitations in mind, how about we look at the simplest possible scenario?

$$u(x,y) = 0 \quad on \quad \partial\Omega \tag{2.2}$$

This is the ongoing challenge that has to be addressed by us. This equation is discretized using techniques such as finite differences, for example. (see [3]). We use an $(n+1)\times(n+1)$ grid on Ω (it means on the unit square), where $\frac{1}{n+1}$ is the grid spacing. Let's denote u_{ij} the approximate solution at $x = ih$ and $y = ih$. This is shown in figure 1, for $n = 7$.

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

$$4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j-1} = b_{ij} \tag{2.3}$$

for all $1 \leq i, j \leq n$. Equation (3) is a linear system when represented in matrix form of $N = n^2$ equations with N unknowns, let's write it

$$A \cdot u = b \tag{2.4}$$

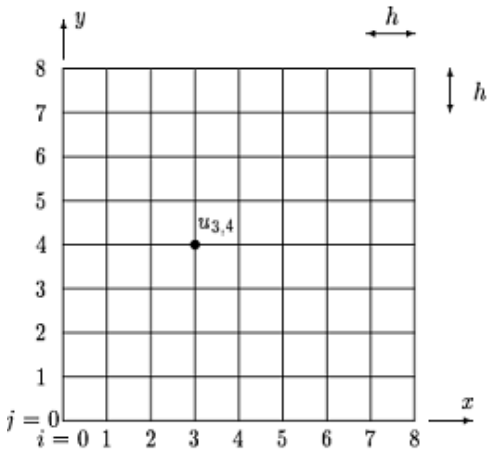


Figure 1: The discrtized doma in $\bar{\Omega}$ with $n = 7$

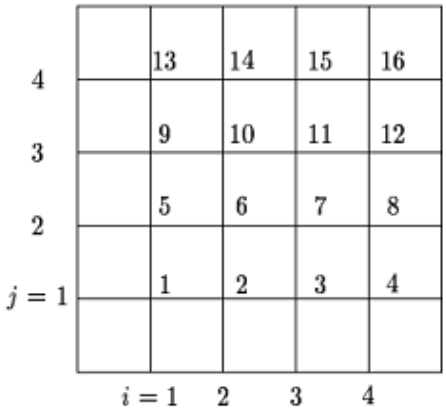
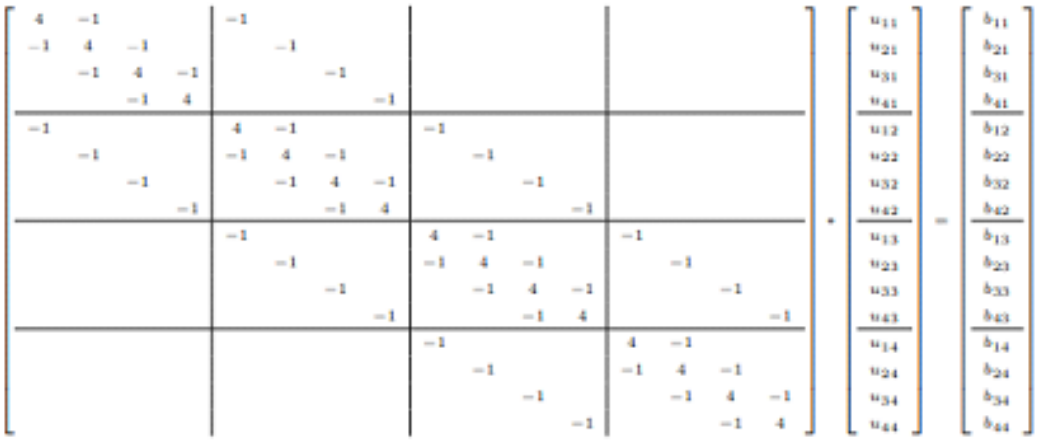


Figure 2: Linerized order of unknowns on a 2D grid

When the linearized order of unknowns is applied to a two-dimensional grid, as shown in figure 2, the system will be as follows:



3 Classical Relaxation Schemes

Jacobi, Gauss-Seidel, and SOR techniques are all included in this discussion. These approaches are derived from (3) in the following way: denoting two subsequent iterations by the numbers k and $k+1$, respectively:

$$u_{ij}^{(k+1)} = \frac{\left(u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + b_{ij}\right)}{4} \tag{3.1}$$

$$u_{ij}^{(k+1)} = \frac{\left(u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + b_{ij}\right)}{4} \tag{3.2}$$

$$u_{ij}^{(k+1)} = \frac{u_{ij}^{(k)} + \omega\left(u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} + b_{ij} - 4u_{ij}^{(k)}\right)}{4} \tag{3.3}$$

(with $1 < \omega < 2$ to have over-relaxation). When using a lattice connection across processors, the article [4] provides parallel alternatives to these approaches that may be applied. A comparison of the computing difficulties of each of these systems is carried out, which may be summarized in the table that follows:

p = number of processors

f = time per flop

α = startup for a message

β = time per word in a message

Complexity (Jacobi) = number of steps $*$ cost per step

$= O(N) * ((N/p)f + \alpha + (n/p)\beta)$

$= O(N2/p)f + O(N)\alpha + O(N32/p)\beta$

Complexity (SOR) = number of steps $*$ cost per step
$= O(\sqrt{N}) + ((N/p)f + \alpha + (n/p)\beta)$
$= O(N32/p)f + O(\sqrt{N})\alpha + O(N/p)\beta$

Remark 3.1. *The Gauss-Seidel technique converges twice as quickly as the Jacobi method, but it needs twice as many parallel steps, utilizing the "checkerboard" ordering of nodes (see [4], [2], and [3]), thus in reality, it takes approximately the same amount of time to execute. Because of this, it does not show in the table that we just looked at. In [3], we demonstrate that the cost per step for the SOR method, as well as the overall complexity of the SOR parallel method, can be reduced cheaper by modifying the connection among processors and employing a ring communication. This is accomplished by using a ring communication.*

4 The Multigrid Methods

It is known (see, e.g. [1,5]) divide-and-conquer is a method for solving discrete problems, and multigrid is an implementation of that algorithm. It is also used extensively in the field of partial differential equations. It's a classic case of divide and conquer in two senses that are connected. To begin, it generates an initial solution for a $(n \times n)$ grid by 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.

If this is the case, the work that we undertake on a certain grid will correct the problem in fifty percent of the frequency components that have not been corrected on other grids. Consider a (2m 1) (2m 1) grid of unknowns. After adding the nodes at the border, which have the value 0, one obtains a $(2m+1)(2m+1)$ grid on which the algorithm will work. This is done without sacrificing generality. Let's notate that n equals $2m$ plus one. Also, we'll use the notation $P(i)$ to refer to the challenge of solving Poisson's equation on a grid with the dimensions $(2i+1)$ by $(2i+1)$ and the unknowns $2i$ 1 by $2i$ 1. The issue is defined by the grid size, which is denoted by i, the coefficient matrix, which is denoted by Ai, and the right hand side, which is denoted by bi. $P(m), P(m1), ..., P(1)$ are a series of linked problems that are formed, where the solution to $P(i$ 1) is a reasonable approximation to the solution of $P(i)$. These problems are generated

on grids that are progressively coarser. Several grids with $n = 9$ are shown in Figure 3. If we designate bi as the right-hand side of the linear system $P(i)$, and xi as an approximate solution of $P(i)$ (meaning that both xi and bi are arrays of values at each grid point with dimensions of $(2i\ 1)(2i\ 1)$), then the Multigrid V-cycle (MGV) may be written as follows:

function MGV (bi, xi) {return an improved solution} xi to $P(i)$

if $i = 1$ {only one unknown}

compute the exact solution $x1$ on $P(1)$

return $(b1, x1)$

else

$xi = Si(bi, xi)$ {improve the solution}

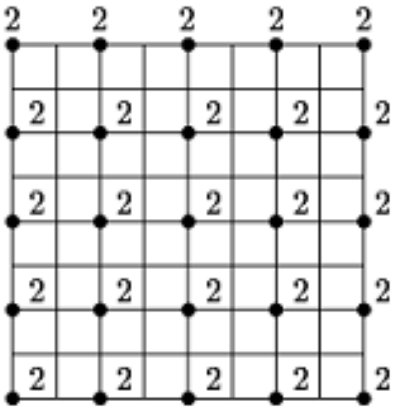
$(bi, di) = Ii - 1(MGV(Ri(bi, xi)))$ {solve recursively}

$xi = xi - di$ {correct fine grid solution}

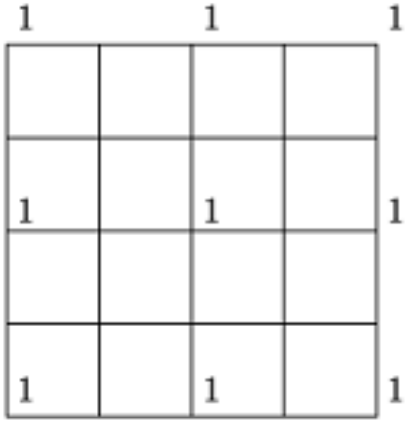
$xi = Si(bi, xi)$ {improve solution some more}

return (bi, xi)

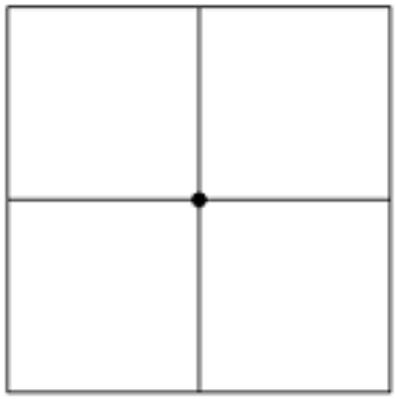
endif



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

Remark 4.1. *Soothing operator (see, for example, [1]), which, in the end, is one or more relaxation stages; Si signifies the smoothing operator. Ri is the restriction operator, which maps the approximate solution for $P(i)$ to the next coarser grid. $Ii1$ is the prolongation operator, which takes an approximate solution $xi1$ for $P(i1)$ and converts it to an approximate on the next finer grid. di is the defect, which indicates how much the solution xi fails in verifying the system.*

Remark 4.2. *If we were to depict the algorithm graphically in space and time (using grid number i), with a point for each recursive call to MGV, it would resemble somewhat like figure 4, beginning with a call to MGV $(P(5), x5)$ in the top left corner. This is the reason why the*

algorithm is referred to as the V-cycle. Because of this, MGV is called on grid 4, then 3, and so on down to the coarsest grid 1, and then back up to grid 5 once again.

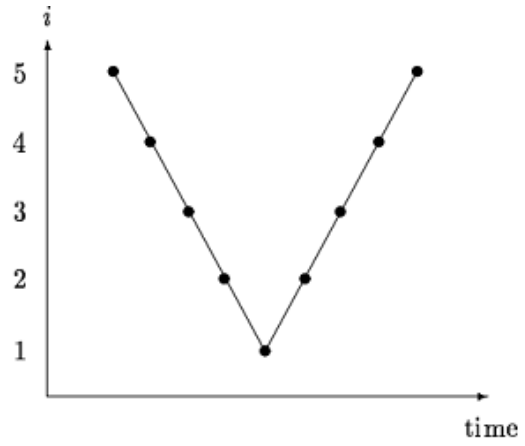


Figure 3: Multi Grid V- Cycle On 5 Grids

The complexity of MGV may be calculated, expressed in terms of big-O notation, if the algorithm is carried out on a serial computer (that is, one with just one processor). Oh, also in this way: we see that the amount of work done at each point in the algorithm is proportional to the number of unknowns, given that the value at each grid point is simply averaged with the values of its closest neighbors. In other words, the work is related to the number of unknowns. Therefore, the cost to reach each point on the "V" in the V-cycle at grid level i is going to be $(2i1)2$, which is on the order of $O(4i)$ operations. If the most precise grid is at level m, the geometric sum will be used to calculate the overall amount of serial labor.

$$\sum_{i=1}^{m}(2^i-1)^2, \quad \text{which is of order} \quad O(4^m) \tag{4.1}$$

Therefore, the total amount of effort in the serial format is directly proportional to the number of unknowns. In a broad sense, has the order $O(N)$, where N is equal to $2n$.

Remark 4.3. *In [4], the Full Multigrid algorithm is also investigated. This method makes use of the Multigrid V-cycle as a fundamental building piece. Due to the fact that it has been shown to have the same level of serial complexity as the Multigrid V-cycle, we will not be insisting on it here.*

5 Conclusions

When looking at the difficulty of the Jacobi, Gauss-Seidel, SOR, and multigrid techniques for solving discrete Poisson's equation on a $n \times n$ grid with $N = 2n$ unknowns, we find that the multigrid approach is the most effective solution. But in addition to that, there are further iterative approaches (of the Krylov type, such as FFT) that provide the same favorable times of execution.

References

[1] W. Briggs, *A Multigrid tutorial*, SIAM, (2008).

[2] I. Chiorean, *On some 2D Parallel Relaxation Schemes*, Burg-Verlag, (2008).

[3] I. Chiorean, *On the complexity of some relaxation schemes*, Buletinul stiintific al Universitatii Baia Mare, Seria B, 18(2)(2002), 171-176.

[4] J. Demmel, *Lecture Notes on Numerical Linear Algebra*, Berkeley Lecture Notes in Mathematics, UC Berkeley, (2012).

[5] W. Hackbusch, *An Introduction to Multigrid Method*, Springer Verlag, (2010).

[6] V. Kumar, *Introduction to Parallel Computing*, The Benjamin Cumming Pub. Company, (2013).

[7] J. J. Modi, *Parallel Algorithms and Matrix Computation*, Oxford, (2012).