Available online at
http://www.ibnusina.utm.my/jfs

Journal of Fundamental Sciences

Article

The parallelization of the Keller box method on heterogeneous cluster of workstations

Norhafiza Hamzah*, Norma Alias, Norsarahaida S.Amin

Department of Mathematics, Faculty of Science, Universiti Teknologi Malaysia.
**To whom correspondence should be addressed. E-mail: norhafizahamzah@gmail.com*

Received 12 April 2008
http://dx.doi.org/10.11113/mjfas.v4n1.34

ABSTRACT

High performance computing is the branch of parallel computing dealing with very large problems and large parallel computers that can solve those problems in a reasonable amount of time. This paper will describe the parallelization of the Keller-box method using the high performance computing on heterogeneous cluster of workstations. The problem statement is based on the equation of boundary-layer flow due to a moving flat plate. The objective is to develop the parallel algorithm of the Keller-box method in purpose to solve a large size of matrix. The parallelization is based on the domain decomposition, where the upper and lower matrices will be splitting into a number of blocks, which then will be compute concurrently on the parallel computers. The experiment was run using 200, 2000, and 20000 size of matrices and using 10 number of processors. The comparison was made from the results obtained from that various size of matrices by doing the analysis based on the performance measurement in terms of time execution, speedup, and effectiveness.

| Keller-box | Parallel algorithm | Parallel computing |

1.Introduction

The box method reported by Keller (1970) and also known as Keller-box method has become popular for obtaining nonsimilar solutions for boundary layer problems [1]. Parallel computing is a form of computing in which many instructions are carried out simultaneously [2]. The problem statement of this paper is based on the equation of boundary-layer flow due to a moving flat plate. The boundary layer theory is often the case for streamlined bodies that these layers are extremely thin, so we can neglect them entirely in computing the irrotational main flow. Once the irrotational flow has been established, we can then compute the boundary layer thickness and velocity profile in the boundary layer, by first finding the pressure distributions evaluated from irrotational flow theory. Then, the results can be used to evaluate the boundary layer flow. The parallel algorithm is implemented to the tridiagonal matrix obtained after the calculation using the Keller-Box method. The parallel algorithm is based on the block LU decomposition.

2. Materials and Methods

The basic idea of the Keller-box method is to write the governing system of equations in the form of a first-order system [3]. To get finite different equations with a second order truncation error, simple centered-difference derivatives and average of the midpoints of net rectangles is used. The resulting finite difference equations are nonlinear algebraic equations. We write the differential equations in finite difference forms first and then linearize the resulting nonlinear algebraic equations by Newton's method. Then, a block-tridiagonal factorization scheme is applied on the coefficient matrix of the finite-difference equations. The governing equations used in this paper are based on the boundary layer equation. In this paper, we will only focus on the block tridiagonal matrix. The matrix is obtained after we applied the finite difference scheme and Newton's method on the boundary layer equation as below:

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=v\frac{\partial^2 u}{\partial y^2} \tag{1}$$

A block tridiagonal matrix is a block matrix, which is having square matrices (blocks) in the lower, main, and upper diagonal, where all other blocks is a zero matrices. It is basically a tridiagonal matrix but has submatrices in places of scalars. A block tridiagonal matrix in our case study has the form as follow:

$$\begin{bmatrix} [A_1] & [C_1] & & & & & \\ [B_2] & [A_2] & [C_2] & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & & [B_{J-1}] & [A_{J-1}] & [C_{J-1}] & \\ & & & & [B_J] & [A_J] \end{bmatrix} \begin{bmatrix} [\delta_1] \\ [\delta_2] \\ \vdots \\ [\delta_{J-1}] \\ [\delta_J] \end{bmatrix} = \begin{bmatrix} [r_1] \\ [r_2] \\ \vdots \\ [r_{J-1}] \\ [r_J] \end{bmatrix}$$

That is : $[A][\delta]=[r]$ (2)

To solve equation (2), we use LU factorization for decomposing *A* into a product of a lower triangular matrix, *L* and an upper triangular matrix, *U* as follows,

$$[A]=[L][U] \tag{3}$$

where

$$[L]=\begin{bmatrix} [\alpha_1] & & & & \\ [B_2] & [\alpha_2] & & & \\ & \ddots & & & \\ & & \ddots & [\alpha_{J-1}] & \\ & & & [B_J] & [\alpha_J] \end{bmatrix} \text{ and } [U]=\begin{bmatrix} [I] & [\Gamma_1] & & & \\ & [I] & [\Gamma_2] & & \\ & & \ddots & \ddots & \\ & & & [I] & [\Gamma_{J-1}] \\ & & & & [I] \end{bmatrix},$$

$[I]$ is the identity matrix of order 3 and $[\alpha_i]$, and $[\Gamma_1]$ are 3x3 matrices which elements are determined by the following equations:

$$[\alpha_1]=[A_1], \tag{4}$$

$$[A_1][\Gamma_1]=[C_1] \tag{5}$$

$$[\alpha_j]=[A_j]-[B_j][\Gamma_{j-1}], \qquad j=2,3,\ldots,J \tag{6}$$

$$[\alpha_j][\Gamma_j]=[C_j], \qquad j=2,3,\ldots,J\text{-}1. \tag{7}$$

Equation (3) can now be substituted into equation (2), and so we get

$$[L][U][\delta]=[r]. \tag{8}$$

If we define

$$[U][\delta]=[W], \tag{9}$$

Then equation (8) becomes

$$[L][W]=[r], \tag{10}$$

The elements *[W]* can be solved from equation (9)

$$[\alpha_1][W_1]=[r_1], \tag{11}$$

$$[\alpha_j][W_j]=[r_j]-[B_j][W_{j-1}], \qquad 2\le j\le J. \tag{12}$$

The step in which Γ_j, α_j and W_j are calculated is usually referred to as the forward sweep. Once the elements of *W* are found, equation (8) then gives the solution δ in the so-called backward sweep. Once the elements of δ are found, Newton's method can be used to find the $(i+1)_{th}$ iteration. These calculations are repeated until some convergence criterion is satisfied and calculations are stopped when $\left|\delta v_0^{(i)}\right| < \varepsilon_1$ where ε_1 is a small prescribed value. In this paper, the value of ε_1 is 0.003.

The parallel implementation is focused on the data decomposition for the equation (3). The data decomposition was design as Figure 1.

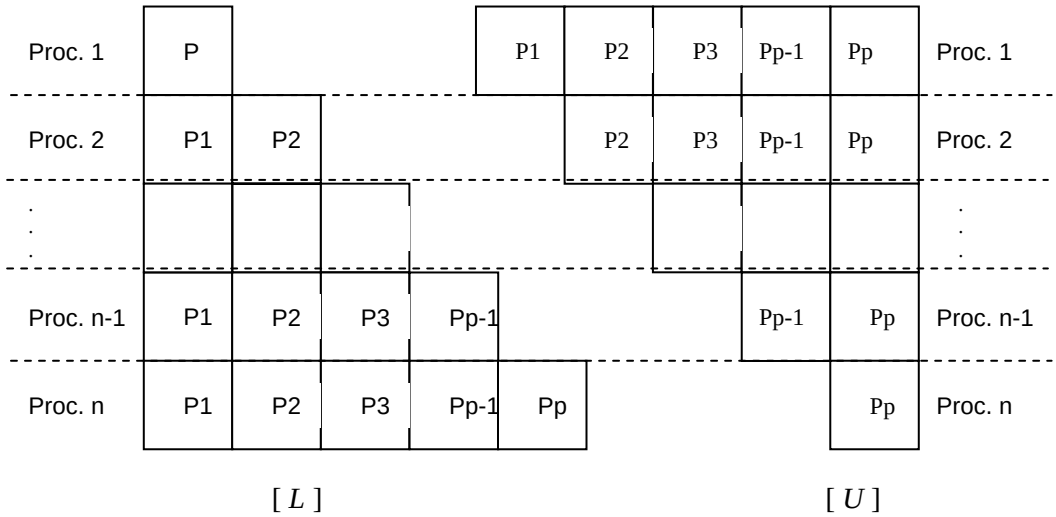

Fig. 1 Data decomposition for calculating L and U matrices.

The data is divided by rows. The P1 to Pp are the processes involved in the parallel implementation where P1 will be calculated by the processor 1 (Proc.1) and after the result of $[W_1]$ from the equation (11) is obtained, it will pass the result to the next processors (Proc.2,…,Proc.n). P2 is solved by the Proc.2 and after it get the result, it will pass the result to the next processors (Proc.3,…,Proc.n) and the last processor, Proc.n will get the final result which then will be used to calculate the equation (9) using the backward sweep.
The computation in the parallel environment can be described as Figure 2 below.

```
program:
...
data = data_sum/no_proc
if CPU="a" then
low_limit=1
upper_limit=50
else if CPU="b" then
low_limit=51
upper_limit=100
end if
do   i   =   low_limit   ,
upper_limit
Task on d(i)
end do
...
end program
```

Fig. 2 Data parallelism pseudocode

Figure 2 shows the pseudocode on how the data is divided to each processor. The pseudocode shows in case if we have two processors, ‘a’ and ‘b’. The difference between *low_limit* and *upper_limit* is the sum of data given to that processor, as in this case a 100 of data is divided by two, so each processor performs the same task on

different pieces of distributed data [4]. Basically the data given to each processor will be calculated by the master processor. The master will divide the data equally and send the data to each worker processor. As in this paper, we study on 200, 2000, and 20000 size of matrix. So to get the size of the data given to each processor, the size of matrix will be divided by 1,2,…,10 number of processors.

3. Results and Discussion

The parallel performance analysis is used to prove parallel algorithm is significantly better than the sequential algorithms. The measurement is done in terms of execution time, speedup, and effectiveness.

The execution time is basically the CPU running time during the calculation of the program in micro second. The bigger size of matrices lead to higher calculation complexity that implies the longer time it takes to execute the process.

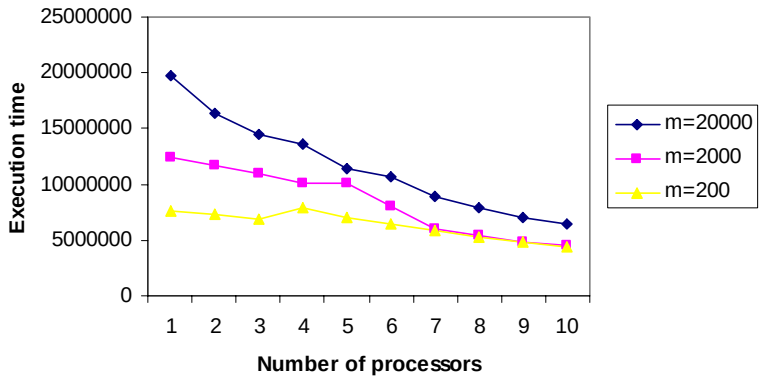

Fig. 3 The execution time vs number of processors

The graph in Figure 3 shows the execution time decreases as the number of processors increases for all size of matrices. The improvement of the performance using the bigger cluster of workstations can clearly be seen in terms of speedup. The execution time graph shows is in micro second.

3.2 Speedup

The Amdahl's law state that the speed of a program is the time to execute the program while speedup is defined as the time it takes a program to execute in serial (with one processor) divided by the time it takes to execute in parallel (with many processors) [5]. The formula of speedup for a parallel application is given as

$$speedup(p) = \frac{Time(1)}{Time(p)},$$

where $Time(1)$ = execution time for a single processor and

$Time(p)$ = execution time using p parallel processors.

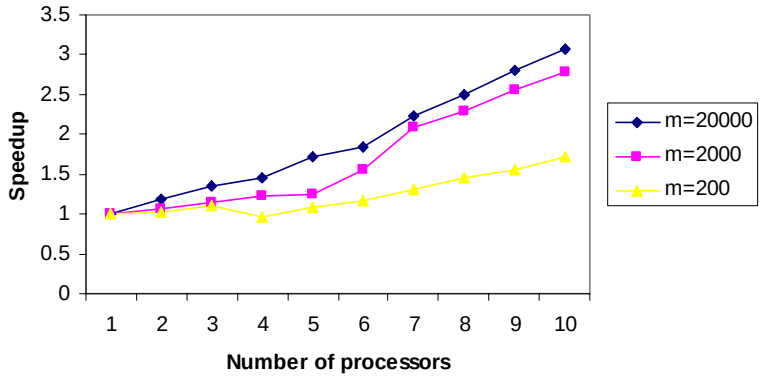

Fig. 4 The speedup vs number of processors

Figure 4 show the speedup for the parallel algorithm. The speedup graph is increasing as the number of processors, p increase. It is because the distributed memory hierarchy reduces the time consuming access to a cluster of workstations. According to Amdahl's Law, the speedup increases with the number of processors increase up to the certain level. From the graph, the speedup with 1 to 5 number of processors increase slower than the speedup of higher number of processors. This is because the more processors we use in the systems, involves more communication and idle time in the process. However, the parallel computing is proved to be most suitable for large sparse matrix problem, since the speedup is at the highest rate for matrix m = 20000.

Effectiveness is used to calculate the speedup and the efficiency. The effectiveness is

$$Effectiveness = \frac{Speedup}{pTime(t)},$$

where

p = number of processors.
$Time(t)$ = execution time using p parallel processors.

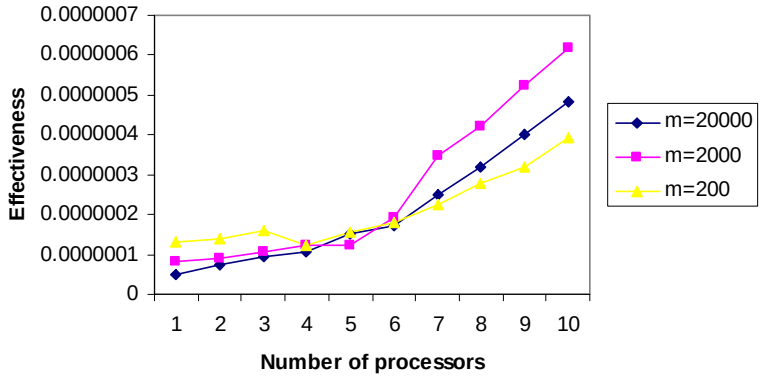

Fig. 5 The effectiveness vs number of processors

Figure 5 shows that the effectiveness increase when the number of processors increasing. The formula of the effectiveness is depends on the speedup, where the speedup increases, the effectiveness will be also increase.

4. Conclusion

Based on the parallel performance analysis, we can see when the number of processors increase the speedup and effectiveness will also be increase, while the execution time is decrease. We can also conclude that communication and computing times is always affected the speedup, and the effectiveness.

5. Acknowledgements

The authors acknowledge the Ministry of Science, Technology and Innovation Malaysia for the financial support through SAGA funding and NSF scholarship.

6. References

[1] E. Jones, Journal of Computational Physics, 40 (1981) 411-429.
[2] G. S. Almasi, and A. Gottlieb, "Highly Parallel Computing", Benjamin-Cummings publishers, Redwood city, CA ,1989.
[3] M. Zuki, "Mathematical Models For The Boundary Layer Flow Due To A Moving Flat Plate",Universiti Teknologi Malaysia,2004.
[4] W. Hillis, Daniel and Guy L. Steele, " Data Parallel Algorithms Communications of the ACM", 1986.
[5] G. Amdahl, Proceedings of the AFIPS Conference, 30 (1967) 483-485.
[6] Norhafiza Hamzah, Norma Alias, Norsarahaida S.Amin, proceedings of the Simposium Kebangsaan matematik & Masyarakat,(2008).