การประชุมวิชาการเครือข่ายวิศวกรรมเครื่องกลแห่งประเทศไทยครั้งที่ 17
15-17 ตุลาคม 2546 จังหวัดปราจีนบุรี

# Parallel Computing on the Navier-Stokes Solver with the Multigrid Method

Kiattisak Ngiamsoongnirn[*], Ekachai Juntasaro[*], Varangrat Juntasaro[**] and Putchong Uthayopas[***]

[*]School of Mechanical Engineering, Institute of Engineering, Suranaree University of Technology,
Nakhon Ratchasima 30000, Thailand,
Phone: (044)224410-2, Email: Kiatt2000@hotmail.com

[**]Department of Mechanical Engineering, Faculty of Engineering, Kasetsart University, Bangkok 10900, Thailand,
Phone: (02)9428555 ext 1829, Email: ovrsk@ku.ac.th

[***]Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Bangkok 10900, Thailand,
Phone: (02)9428555 ext 1416, Email: pu@ku.ac.th

## Abstract

This paper is aimed to present the combination of the parallel computing and the multigrid method on the Navier-Stokes solver. The combination is based on the concept of the object-oriented programming (OOP), which consists of 4 independent modules: Grid Generation, Navier-Stokes Solver, Multigrid Method and Parallel Computing modules. The multigrid method is implemented by employing the full approximation storage (FAS) scheme for numerically solving the non-linear Navier-Stokes equations. The overall computation is performed by using the parallel computing in which a number of computers are concurrently computed for the same task but on different sub-data. The two-dimensional laminar flow in a cavity at Re=1,000 is used as a test case. It is found that the computational time is decreased significantly when employing the combination of the multigrid method and the parallel computing.

## 1. Introduction

In recent years, Computational Fluid Dynamics (CFD) has been a design tool in industries due to, for example, the lack of instruments to measure some quantities in some dangerous zones. In addition, the advantages of CFD are the low cost to construct and the ability to immediately observe some phenomena through a monitor of personal computer (PC). However, when the flow is so complicated, the number of data points required to capture the physics of flow has to be large enough. A single computer is limited to its memory and speed. In other words, it may compute one problem for a large number of iterations or it cannot handle with a large number of data at all. The convergence rate of iterative methods can be greatly improved by using multigrid acceleration techniques. The multigrid methods eliminate some errors on the coarser grid that cannot be eliminated by the finer grid and then correct the solutions up to the finest grid. The time consumption, moreover, can be decreased by using a large number of computers to simultaneously solve the same problem, which is the so-called parallel computing. The data is partitioned to the smaller data and assigned to each processor for performing the same function. This kind of parallel computing is called the Single Instruction Multiple Data (SIMD) architecture

In this paper the combination of the parallel computing and the multigrid method on the Navier-Stoke solver is presented. The solver code is carried out for the steady laminar flow in a two-dimensional cavity. The numerical solutions are then compared with Ghia, Ghia, and Shin [1]. To take into account the data synchronization, interchanging and updating the boundary data at the interface between a pair of adjacent processors are essential. Therefore, the Message-Passing Interface (MPI) library is used in the present work to perform such a task.

## 2. Governing Equations

The present work is focused on the incompressible laminar flow which is governed by the continuity and Navier-Stokes equations where all the fluid properties are treated to be constant.

### 2.1 Continuity Equation

The continuity equation is derived from one of the basic laws of physics: if there is no mass generated in the control volume, the amount of mass entering the control volume has to be equal to the amount of mass passing out of the control volume.

$$\frac{\partial}{\partial x_j}\left(\rho u_j\right)=0 \qquad (1)$$

where $\rho$ is the fluid density and $u_j$ are the velocity components.

### 2.2 Navier-Stokes Equations

The second law of Newton is all about the conservation law of momentum. Applying the conservation law of momentum over a fluid particle and treating it as the Newtonian fluid, the Navier-Stokes equations for the steady incompressible laminar flow can be derived in the form of tensor

$$\frac{\partial}{\partial x_j}(\rho u_j u_i)=\frac{\partial}{\partial x_j}\left(\mu\frac{\partial u_i}{\partial x_j}\right)-\frac{\partial p}{\partial x_i} \qquad (2)$$

where $\mu$ is the viscosity and $p$ is the pressure.

## 3. Numerical Method

In this work, the finite volume method is employed to numerically solve the governing equations which consist of the continuity and Navier-Stokes equations. The numbering system E, W, N, and S are referred to East, West, North, and South respectively.

Integrating the continuity equation over the control volume gives

$$F_e-F_w+F_n-F_s=0 \qquad (3)$$

where $F_e=(\rho u)_e$, $F_w=(\rho u)_w$, $F_n=(\rho v)_n$, and $F_s=(\rho v)_s$.

Integrating the Navier-Stokes equations over the control volume with the second-order central differencing scheme for the diffusive terms and with the first-order upwind scheme for the convective terms, the standard form of the finite volume equation can be arranged as

$$a_P\phi_P=a_E\phi_E+a_W\phi_W+a_N\phi_N+a_S\phi_S+S \qquad (4)$$

where $a_E=D_e+\max(0,-F_e)$,

$a_W=D_w+\max(F_w,0)$,

$a_N=D_n+\max(0,-F_n)$,

$a_S=D_s+\max(F_s,0)$

$a_P=a_E+a_W+a_N+a_S$

$S=-\left(\frac{\partial p}{\partial x_i}\right)_P dV$

and $\phi=u$ *and* $v$ (streamwise and cross-stream velocities respectively).

Once the continuity and Navier-Stokes equations have been discretised, the Semi-Implicit Method for Pressure-Linked Equation (SIMPLE) is employed to solve these discretised equations in order to avoid the decoupling problem between velocity and pressure fields. The collocated grid arrangement is used in this work so that all variables are stored at the center of each control volume. The Rhie-Chow interpolation is the process of determining the mass flux that entering into and passing out of the control volume to take into account for the nonlinear pressure However, in the present work, the Rhie-Chow interpolation is not used. The SIMPLE algorithm [2] for the simulation of steady incompressible laminar flow can be summarized as follows:

(1) Initialize the velocity and pressure with the guessed values.
(2) Solve the Navier-Stokes equations for the velocity field.
(3) Solve the pressure correction equation for $p'$.
(4) Correct the pressure and velocity by the pressure correction $p'$.
(5) Repeat steps (2)-(4) until the solution converges.

## 4. Multigrid Method

One drawback of conventional iterative methods is that they cannot eliminate the low-frequency component of error effectively. However, the low-frequency error on the fine grid appears as the high-frequency error on the coarser grid [3]. Therefore, it is worthwhile to eliminate the high-frequency error on the coarser and coarser grid and then evaluate the corrections to correct all

the way up to the next finer and finer grid solutions. This strategy is called the multigrid technique.

The multigrid method is the combination of the smoothing process, in which the equations on each grid level are solved, and the restriction and prolongation processes, which transfer the current approximate solutions, equation residuals and correction quantities between adjacent grid levels: the restriction transfers the data down to the coarser grid and the prolongation transfers the data up to the finer grid. The multigrid scheme specifies how the coarse-grid problem is generated from the fine-grid problem and what order the multiple grid levels are visited, i.e., the cycle types, for example, the multigrid V-cycle is the one where a recursive algorithm has the following steps: pre-smoothing iterations, restriction to a coarser grid, solving the coarse grid problem for coarse grid corrections, prolongation of corrections to the fine grid, and post-smoothing iterations. The problems are solved up and down in the same fashion which leads to the V shape of the cycle, as shown in Fig. 1, the number of pre- and post-smoothing iterations are specified inside the circles. The number of iterations is fixed for any appropriate value and is the same in the pre- and post-smoothing iterations but may be different if the optimization is required.

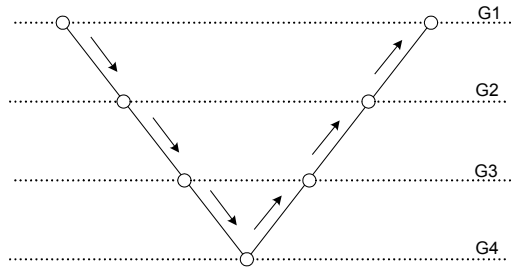


**Fig. 1** Multigrid V-Cycle

In the case of linear problems, a multigrid correction scheme (CS) can be used effectively because of the solution errors are directly proportional to the solution residuals. Therefore, only the solution residuals are restricted to the next coarser grid to solve the coarse grid problem for evaluating the corrections. For nonlinear problems, however, the CS treatment cannot be employed. For example, consider a system of nonlinear algebraic equations, **A(u) = f**, where **A** is the matrix coefficient of vector **u** and **f** is the vector of source term. Suppose that **v** is an approximation to the exact solution **u**. If the solution error is simply **e = u – v** and the residual is **r = f – A(v)**. Substract the original equation from the definition of the residual to give **A(u) – A(v) = r.** Applying the nonlinear operator to the definition of the error which can be written as **A(e) = A(u - v)**. Since **A** is nonlinear, the vector **u** and **v** cannot be easily split out of the operator **A**, hence **A(u - v) ≠ A(u) - A(v).** Moreover, the relation **A(e) = A(u) – A(v) = r** is no longer valid unlike linear problems. Therefore, there is no simple linear residual equation and thus the correction scheme cannot be used to solve the nonlinear problems.

As stated earlier, the linear equation is possible to “transfer the problem” from one grid to another by merely transferring the residual. If the equation is nonlinear, the transfer of residual alone is generally not possible [3]. Thus, in the case of nonlinear equation, the solution must be transferred together with the residual. This version of multigrid method is known as the *full approximation storage* (FAS) scheme. The detailed derivation of the FAS can be found in [4], this paper illustrates only three levels of the FAS algorithm in which there are the following steps:

- Solve the finest-grid problem, $A^h(v^h) = f^h$, and then restrict the current approximation and its residual to the intermediate coarser grid: $r^{2h} = I_h^{2h}(f^h - A^h(v^h))$ and $v^{2h} = I_h^{2h} v^h$ .
- Solve the intermediate-grid problem, $A^{2h}(v^{2h}) = f^{2h}$, and then restrict the current approximation and its residual to the coarsest grid: $r^{3h} = I_{2h}^{3h}(f^{2h} - A^{2h}(v^{2h}))$ and $v^{3h} = I_{2h}^{3h} v^{2h}$, where $f^{2h} = A^{2h}(I_h^{2h} v^h) + r^{2h}$.
- Solve the coarsest-grid problem, $A^{3h}(v^{3h}) = f^{3h}$, and then compute the coarsest-grid approximation to the error: $e^{3h} = v^{3h} - I_{2h}^{3h} v^{2h}$, where $f^{3h} = A^{3h}(I_{2h}^{3h} v^{2h}) + r^{3h}$.

Up to now, the restriction process has completed and the further steps are the prolongation process:

- Interpolate the coarsest-grid error approximation up to the intermediate grid and correct the current approximation of such a grid: $v^{2h} = v^{2h} + I_{3h}^{2h} e^{3h}$ .
- Solve the intermediate-grid problem, $A^{2h}(v^{2h}) = f^{2h}$, with the last updated solution as the initial guess and with the same source term as in the restriction process and then compute the intermediate-grid approximation to the error: $e^{2h} = v^{2h} - I_h^{2h} v^h$.
- Interpolate the intermediate-grid error approximation up to the finest grid and correct the current approximation of such a grid: $v^h = v^h + I_{2h}^h e^{2h}$.

To incorporate the multigrid technique with the Navier-Stokes equations, some special treatment must be taken carefully. The velocity components are nonlinear but pressure is linear and both the velocities and pressure appear in the system of equations. Therefore, the pressure is solved through the pressure correction equation by the multigrid correction scheme and the FAS is used

for the solution of the velocity components. The current approximation of the velocity components, u and v, as well as the residual of the momentum and pressure correction equations are then restricted to the next coarser grid. Once a coarse grid has been visited, the coarse-grid pressure is initialized with the guessed value of zero every time, and all the coefficients together with the mass flux must be recalculated on this grid with the restricted solution, and then the process of the SIMPLE algorithm is proceeded for a few iterations with the same steps as in a single-grid problem. As the restriction process has gone down to the coarsest grid and the coarsest-grid problem is solved, the change in the velocity components and the current solution of the pressure correction are then prolonged up to the next finer grid for correcting the fine-grid current approximation of pressure and velocity components: u and v with the change in velocity components that prolonged from a coarse grid; p with the prolonged pressure correction. Thereafter, the same operation as the restriction process is repeated again, but the way is up instead of down and the last updated solution is used as the initial guess as well as the pressure which is initialized to zero on the restriction process, and hence the change in velocity components which is prolonged up to the next finer grid is the difference between the current approximation and the restricted solution at the restriction process. This is the multigrid V-cycle and one cycle has been completed as the process goes up to the finest grid, the bilinear interpolation is used to restrict the velocity components or prolong the corrections and the residuals are restricted by simply summing the residuals of the four fine grid control volumes that make up each coarse grid control volume.

## 5. Parallelization Method

The basic idea of the parallel computing is that a number of processors work in cooperation on a single task. For a distributed memory architecture each processor possesses its own memory and connects to another ones through the high-speed interconnection network. This is called a cluster of computers. Each processor can access only its own memory space. Therefore, sending data into and receiving data from other processors are carried out by means of a message-passing interface. The message passing is done by using the MPI library, which is portable to most computers [5]. The strategy of parallel computing can be classified into two categories: data parallelization and task, or function, parallelization. The data parallelization is to partition the overall data into a number of sub-data. The number of sub-data is equal to the number of processors. Each sub-data is assigned to each processor where each processor performs the same task or the same code on its own sub-data. This strategy is called the Single Instruction Multiple Data architecture (SIMD). The other is the functional parallelization on the Multiple Instruction Multiple Data (MIMD) architecture, which assigns different tasks or functions to different data on different processors. In this case, each processor possesses the same amount of data (or can be different) and this data is performed with the function that is different from the functions that are performed by other processors.

In the present work, the data parallel computing is chosen because it is more appropriate for the present CFD algorithm than the functional parallel computing. This is because the functions used in the present algorithm are solved simultaneously. It should be noted that if two computers possess the same name of variable, and if the variable in the first computer is changed, the second computer will not sense that changing as long as the message is not sent to inform and exchange such a variable. Thus, if the CFD code in the present work is operated on the functional parallel computing, the message has to be sent every time for every calculated data point. This gives rise to the time overhead that increases the time consumption.

### 5.1 Parallel Algorithm

The parallel computing model should be discussed before considering the parallel algorithm. In this paper the model of parallel computing is the master-slave model [6]. The master is assigned to initialize the necessary data such as boundary and initial conditions and send this data to all slaves and the master itself. The slaves receive the corresponding data from the master and each slave carries out its own data with the same code as the sequential algorithm does (see Fig. 2).

The algorithm of the parallel computing used in the present work can be summarized as follows:

*The master*:

1) Determine the number of processors to be used in the calculation and then evaluate the size of each sub-data that is the size of the overall data divided by the number of processors.
2) Allocate the memory space for the overall data and for each sub-data.
3) Specify the physical boundary and initial conditions on the overall data.
4) Send the initialized overall data to the corresponding slave sub-data.

5) Perform the solver code on each sub-data until the maximum error of all processors reaches the setting criteria. In this step, the error of each processor has to be interchanged among all processors to find the maximum error. In addition, the data at the boundaries of the adjacent processors have to be exchanged to synchronize the data only on the finest grid.

6) Transfer each sub-data directly to its corresponding location in the overall data.

*7)* Display the final results of the overall data (if necessary).

*The slave*:

1) Determine the number of processors to be used in the calculation and then evaluate the size of each sub-data that is the size of the overall data divided by the number of processors.

2) Allocate the memory space for the corresponding sub-data or sub-domain.

3) Receive the corresponding location of the initialized overall data from the master.

4) Perform the solver code on each sub-data until the maximum error of all processors reaches the setting criteria. In this step, the error of each processor has to be interchanged among all processors to find the maximum error. In addition, the data at the boundaries of the adjacent processors have to be exchanged to synchronize the only on the finest grid.

5) Send each sub-data back to the corresponding location in the master overall data.

As all processors have executed the solver code, the sequential steps of all processing nodes, as shown in Fig. 3, are run concurrently. However, some processors may be idle if they have been waiting for the messages from a pair of SEND-RERCIEVE processors that have not sent the data. Since the present work uses the standard blocking send/receive MPI routine, the receiver will not be able to receive the data and has to wait if the sender does not send any. This is the disadvantage of poor design of parallel algorithm.

**5.2 Communication Strategy**

In step (5) of the master-processing node and step (4) of the slave-processing node, the calculation has been carried out on each sub-domain using the same code. In case of the sequential code of two-dimensional problems, there are four physical boundaries. It should be noted that the parallel code is the copy of the sequential code; therefore, each sub-domain also has four boundaries. The exchange of the data at the interface of the pair node is necessary to avoid the fictitious internal boundary.

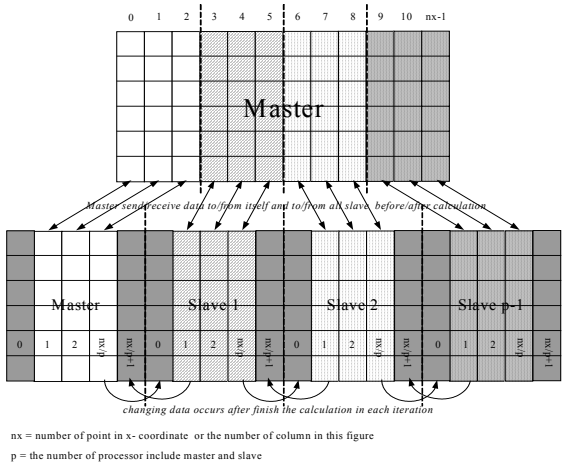


**Fig.2** A model of parallel computing

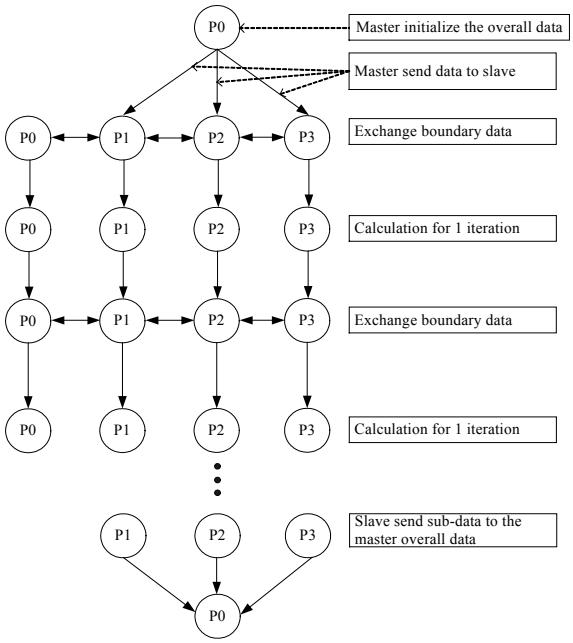


**Fig.3** A sequence of each processor after executing a program

Data allocation for each sub-domain is depicted in Fig. 3. It can be seen that the number of columns for each sub-domain after partitioning is incremented by two columns (the gray columns). These are called the *ghost cells*. It is served as the boundary for each sub-domain in each node. The first location of computing is the second column and now it is no longer the boundary, likewise the last location of computing is not the last column but it is the column before the last column (it is no longer the boundary as well). Let us consider the exchange of the data of the slaves 1 and 2. It occurs after the computing has done per iteration. The first column of the slave 2 is treated as its left

boundary which requires the data from the fourth column of the slave 1. Also the last column of the slave 1 is treated as its right boundary which requires the data from the second column of the slave 2. The others can be interpreted in a similar manner except for the first column of the sub-domain of the master and the last column of the last slave that are left empty. Therefore, it can be seen similar to the sequential code, i.e., the second column in Fig. 3 (number 1) of the sub-domain of the slave 1 which is the fourth column (number 3) of the overall data of the master. It seems as if the data of all slaves treated the same as the data of the primary domain. On the other hand, the sixth and seventh columns (number 5 and 6) of the overall domain of the master, if performed by the sequential code, are updated immediately when the computing is operated. However, the last column of the slave 1 (the right boundary of the slave 1) and the first column of the slave 2 (the left boundary of the slave 2) are updated per iteration. Therefore, they are still the semi-fictitious internal boundary conditions. This causes the solution convergence to slowdown.

## 6. Parallel Multigrid Implementation

There are two approaches to incorporate the parallel computing with the multigrid method [7]: the domain decomposition with multigrid and the multigrid with grid partitioning. The domain decomposition is first applied to the finest grid. Then, the multigrid method is used to solve problems inside each block, that is, the multigrid V-cycle is applied in each local finest grid. Since inter-domain connection is limited to the finest level, thus communications are only required at this level. For the multigrid with grid partitioning, however, the multigrid method is used to solve the problem in the whole grid in which the grid is partitioned among processors at each level. Therefore, communications are required at each level

Domain decomposition methods are often used with the finite element method on parallel computers [8]. They are easier to implement and require fewer communications only on the finest grid. Additionally, they can be applied to general multiblock grids. On the other hand, the domain decomposition methods lead to algorithms which are numerically different from the sequential version and have a negative impact on the convergence rate. Grid partitioning retains the convergence rate of the sequential algorithm. However, it requires more communication overhead because the data exchange at each grid level. Several literatures adopted the grid partitioning technique, for example, [8, 9], but in the present work the domain decomposition is employed.

## 7. Results and Discussion

A lid-driven cavity flow at Re=1000 is used as a test problem. Fluid with u=1.0 m/s and density=1.0 kg/m$^3$ flows past over the top of the cavity as shown in Fig. 4 which depicts the velocity contour. The computed u-velocity along the vertical centerline and the computed v-velocity along the horizontal centerline of a square cavity are compared with the results of Ghia et al [1]. The results are agree very well with the reference data. Fig. 6 shows the computing time used by the multigrid (MG) and the single grid (SG) method, the single grid method uses about 5 times as many as the multigrid method does.

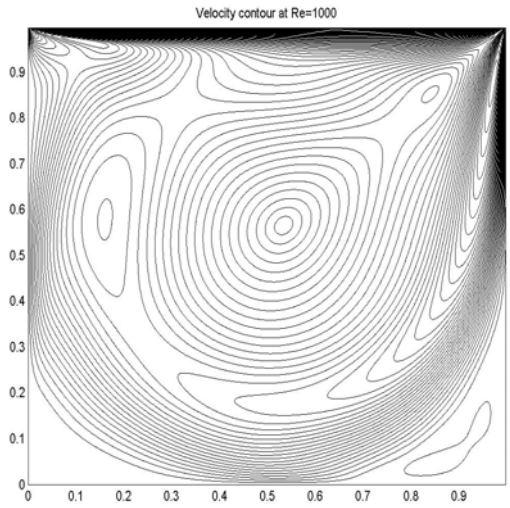


**Fig.4** Velocity contour at Re=1000

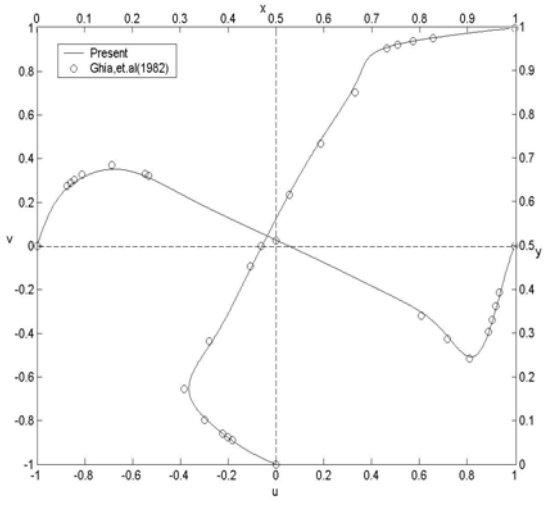


**Fig. 5** u- and v-velocity at Re=1000 are compared with the reference data

In order to test the parallel multigrid scheme, requested for parallel machine, all cases are run and tested on the 8 single-processing nodes of CAMETA cluster located at Computational

Fluid Dynamics Laboratory, Suranaree University of Technology, Nakhon Ratchasima, Thailand. In Fig.7, the residuals are plotted versus the computing time which are used by the multigrid without parallel and the parallel multigrid method with 2, 4 and 8 processors respectively. It is found that the parallel computing with the multigrid method does not improve the overall computation performance. However, with 8 processors the time consumed by parallel computing with the multigrid method is slightly less than a single processor with the multigrid method. This is because the number of grid points is large enough. The multigrid technique is deteriorated in convergence rate when using with the parallel computing because of the effect of the internal boundary. It is observe in Fig. 7 that the line of the parallel computing have strongly fluctuation with respect to time, but the line of the multigrid method without parallel computing is nearly smooth. It can be concluded that the oscillation occurred is the effect of the internal boundary. Moreover, this work employs the domain decomposition with the multigrid. Only the finest grid is taken into account the synchronization of data, which possesses the disadvantage of the rate of convergence. However, the parallel version with 8 processors has overcome such a poor rate of convergence, because the computing time is so fast that the influence of internal boundary is diminished. It is interesting to think further that if the larger the number of grid points and the larger the number of processors are used, how much faster (or slower) the parallel multigrid scheme is compared with the multigrid method without parallel computing, this is left to study further and since the present work is limited only with 8 number of processors(nodes). Table 1 reports the time consumption of the parallel computing with and without the multigrid method until the solution converges. The parallel single grid method is about 7 times slower than the parallel multigrid method.

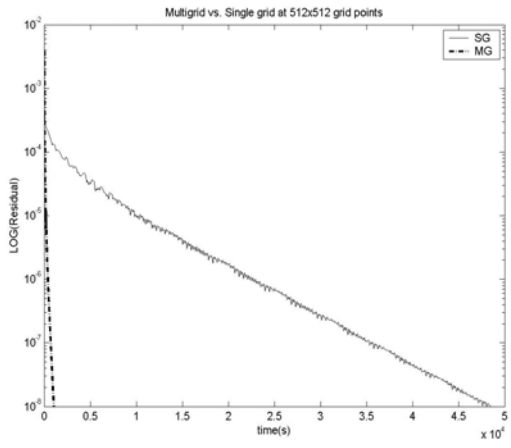


**Fig. 6** Multigrid and single-grid result on a single processor

**Table 1** Comparison of parallel with and without multigrid

| Number of Processors | SG (time in s) | MG (time in s) |
|---|---|---|
| 2 | 2.349704e+04 | 3.338826e+03 |
| 4 | 1.189300e+04 | 1.613995e+03 |
| 8 | 6.338385e+03 | 9.328902e+02 |

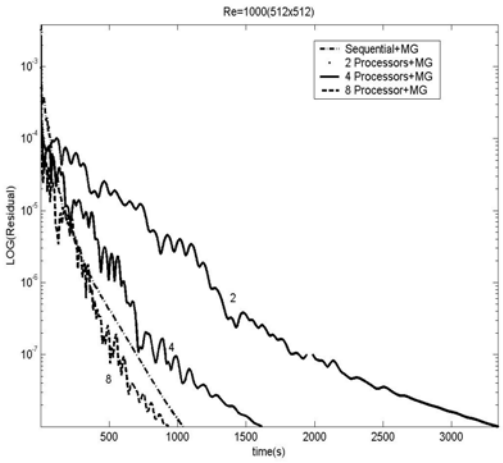


**Fig. 7** Multigrid with and without parallel computing

## 8. Conclusions

The multigrid method with and without the parallel computing and the single grid method with and without the parallel computing are studied. The program is firstly validated by comparing the results at Re=1000 with [1] and very good agreement is found. Without parallel computing, the efficiency of the multigrid technique is estimated by comparing the computing time of the single grid against multigrid, the multigrid technique uses about 5 times less computing time than the single grid. The performance of parallel multigrid is evaluated by comparing the time consumption of the multigrid technique with a single processor versus the single grid parallel computing. With parallel computing, the multigrid method is about 7 times faster than the single gird method. However, the parallel multigrid with 8 processors is slightly faster than the sequential multigrid algorithm. For the parallel multigrid with 2 and 4 processors, the sequential multigrid algorithm is faster than the parallel multigrid. In brief, this present work with limited resource(hardware) and the parallel computing in conjunction with the multigrid technique can improve significantly an overall efficiency. The efficiency can be improved by increasing the number of processing nodes and an enhancement of the parallel multigrid scheme still has been

carrying on until to reach an extreme performance. In the future work, the study will be focused on the 3D turbulence flow simulation and the number of processing nodes is up to 16 nodes and hence the parallel multigrid computation will be improved to acquire a high efficiency as possible as.

## 9. Acknowledgement

The present work is financially supported by the National Electronics and Computer Technology Center (NECTEC). This support is greatly appreciated.

## 10. References

[1] U. Ghia, K. N. Ghia, C. Shin, High-Re Solutions for incompressible Flow using the Navier-Stokes Equation and a Multigrid Method, J. Comp. Physics, Vol. 48, pp. 387-411.

[2] H. K. Versteeg and W. Malalasekera, An Introduction to Computational Fluid Dynamics-The Finite Volume Method, Longman Group,1995 .

[3] D. A. Anderson, J. C. Tannehill. and R. H. Pletcher, Computational Fluid Mechanics and Heat Transfer, Hemisphere Publishing Corporation, Taylor & Francis Group, New York, 1984.

[4] W. L. Briggs, V. E. Henson, and S, F. McCormick, A Multigrid Tutorial 2nd(SIAM)

[5] B. Troff, O. Labbe′, and P. Saguat, Parallel Implementation of DNS Solver, Computational Fluid Dynamics' 98, Proceedings of the Fourth European Computational Fluid Dynamics Conference, Vol. 1, pp. 490-493

[6] T. Callitsis and G. Bergeles, A Multi-Block Method
For Solving Incompressible Viscous Flows, Computational Fluid Dynamics' 98, Proceedings of the Fourth European Computational Fluid Dynamics Conference, Vol. 1, pp. 490-493

[7] M. Prieto, R. S. Montero, and I. M. Llorente, A Parallel Multigrid Solver for Viscous Flows on Anisotropic Structured Grids, Tech. Rep. 2001-34, 2001

[8] M. Prieto, R. Santiago, D. Espadas, I. M. Llorente, and F. Tirado, Parallel Multigrid for Anisotropic Elliptic Equations, J. Parallel and Distributed Computing Vol. 61, pp. 96-114, 2001

[9] J. Z. Lou and R. Ferraro, A Parallel Incompressible Flow Solver Package with a Parallel Multigrid Elliptic Kernel, J. Comp. Physics, Vol. 124, pp. 225-243, 1996