Journal of Science and Technology

Incompressible Flow Simulation Using SIMPLE Method on Parallel Computer

Bukhari Manshoor[1]*, Mat Nawi Wan Hassan[2] and Norma Alias[3]
[1]Faculty of Mechanical Manufacturing and Engineering ,
Universiti Tun Hussein Onn Malaysia
[2]Faculty of Mechanical Engineering,
Universiti Teknologi Malaysia
[3]Faculty of Science,
Universiti Teknologi Malaysia

Corresponding email: bukhari@uthm.edu.my

Abstract

This paper is concerned with the development of a parallel algorithm on cluster of workstation with Parallel Virtual Machine (PVM) for solving the finite difference Navier-Stokes and energy equations. The numerical procedure is based on SIMPLE (Semi Implicit Method for Pressure Link Equations) developed by Spalding. The governing equations are transformed into finite difference forms using the control volume approach. The hybrid scheme which is a combination of the central difference and up wind scheme was used in obtaining a profile assumption for parameter variations between the grids points. The Domain Decomposition Method (DDM) was used to decompose the domain into a small domain and each of the domains was solved by different processors. The accuracy of the parallelization method was done by comparing with a benchmark solution of a standardized problem related to the two dimensional buoyancy flow in a square enclosure. The results are shown in the forms of contour maps of non-dimensional temperature and velocities.

Keywords: SIMPLE algorithm, Parallel Algorithm, Domain Decomposition Method, Navier-Stokes Equations

1 INTRODUCTION

Parallel computing or also known as parallel processing refers to the concept of speeding up the excitation of a program using multiple processors by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. A program being executed across n processors might execute n times faster than it would use a single processor. Here we can say that the parallel processing is differs from multitasking in which a single processor execute several programs at once.

Since a new generation of single processor computer is a costly enterprise in order to obtain a larger and faster communications, parallel computing becomes a key for highperformance architecture. All cotemporary supercomputers are parallel processing computers. Massively Parallel Processors (MPPs) are now the most powerful computer in the world. These machines combine a few hundred to a few thousand CPUs in a single large cabinet connected to hundreds of gigabytes of memory.

This project deals with a development of parallel algorithms in order to solve an incompressible flow simulation using SIMPLE method that originally put forward by Patankar and Spalding (Davis,1983). The analysis of an incompressible flow become more complicated and need a high performance computer to solve the problem. One of the problem during to solve the complicated problem on incompressible flow is time constraint. More complicated of the problem means more time should be spend to solve the problem.

To overcome this problem, parallel computer was used and to determine the performance of this parallel computations, the corresponding parallel algorithms was developed and it based on method of parallelization known as Domain Decompositions Method. As the number of the nonlinear simultaneous equations formed after discretisation of the modelling equations is large, an iterative technique is used to update the flow variables. Control volume approach is selected and the matrix formed will solved using matrix tri-diagonal solver. At the end of this paper, the result of simulation using parallel algorithms are presented and discussed.

2 NUMERICAL ANALYSIS

2.1 Governing equations

Two-dimensional incompressible laminar constant-density flow and energy equation is governed by set of partial differential equations (Melaaen, 1993). The continuity, momentum and energy equations in their primitive form are shown in equations (1-4) where the equation for conservation of mass is given by:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \qquad (1)$$

The conservation of momentum in x and y directions are governed by the u-momentum equation expressed as:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \frac{\mu}{\rho}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + S_u \qquad (2)$$

as well as the v-momentum equation:

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + S_v \qquad (3)$$

The conservation of energy express as:

$$u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \frac{k}{\rho c_P}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \qquad (4)$$

The terms on the left-hand side of equations (2) and (3) are convective terms and the terms on the right-hand side include the pressure gradient and viscous terms. In all the above equations, u and v are the x and y components of the velocity, P is the pressure and ñ is the density.

2.2 Discretisation

In order to numerically solve the velocity and pressure fields that obey the discretized momentum and continuity equations, the finite difference method was applied. This method involves integrating the continuity and momentum equations over a two-dimensional control volume on a staggered differential grid shown in Figure 1 (Patankar et al., 1972).

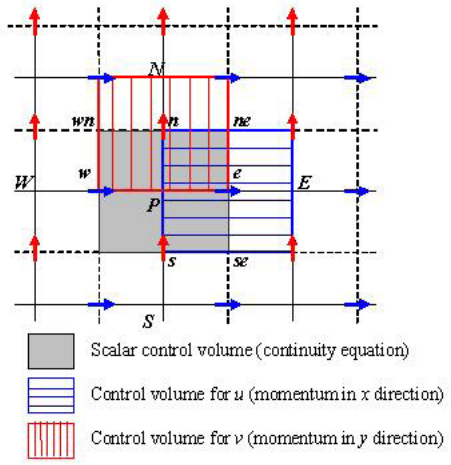

N
wn
n
ne
W
w
e
E
P
s
se
S
Scalar control volume (continuity equation)
Control volume for u (momentum in x direction)
Control volume for v (momentum in y direction)

Figure 1 Staggered Grid showing locations of flow variables (staggered forward).

This yields the governing equations in their discretized form as shown in equations (5-6). The staggered grid evaluates the scalar variables, in this case only the pressure, which are stored at the scalar nodes and located at the intersection of two unbroken grid lines. Such points are indicated by the capital letters P, W, E, N and S. The u-velocity components are stored at the east and west cell faces of the scalar control volume and are indicated by the lower case letters e and w. The v-velocity components are stored at the north and south cell faces of the scalar control volume which are indicated by the lower case letters n and s. These velocity components are located at the intersection of a dashed and unbroken line that construct the scalar cell faces and are indicated by arrows.

The horizontal arrows shown indicate the locations of the ue and uw velocity components and the vertical arrows indicate the locations of the vn and vs velocity components. After the process of discretization, the discretized of u-momentum equation becomes:

$$a_{i,J}u*_{i,J} = \sum a_{nb}u*_{nb} + \left(p*_{I-1,J} - p*_{I,J}\right)A_{i,J} + b_{i,J} \tag{5}$$

and discretized of v-momentum equation becomes:

$$a_{I,j}v*_{I,j} = \sum a_{nb}v*_{nb} + \left(p*_{I,J-1} - p*_{I,J}\right)A_{I,j} + b_{I,j} \tag{6}$$

The method of discretizing the momentum and continuity equations is fully explained by (Malalasekera at al., 1995) where they show how to employ differencing schemes to successfully interpolate between the node points.

3 SOLUTION PROCEDURE OF THE SIMPLE ALGORITHM

The SIMPLE method proceeds by a cyclic series of guess and correct operations. The important operations are described in the following steps below. The flow chart of the algorithm showed in Figure 2.

i. Guess the pressure field, p*.
ii. Solve the momentum equation to obtain u* and v*.
iii. Solve the pressure correction equation to obtain p'.
iv. Calculate p form equation "" by adding p' to p*.
v. Calculate u and v from their starred values using velocity correction equation.
vi. Solve the discretization equation for other ø's (for this case, we solve the energy equation to obtain temperature T)
vii. Treat the corrected pressure p as new guessed p*, return to step 2 and repeat the whole procedure until a converged solution is obtained.

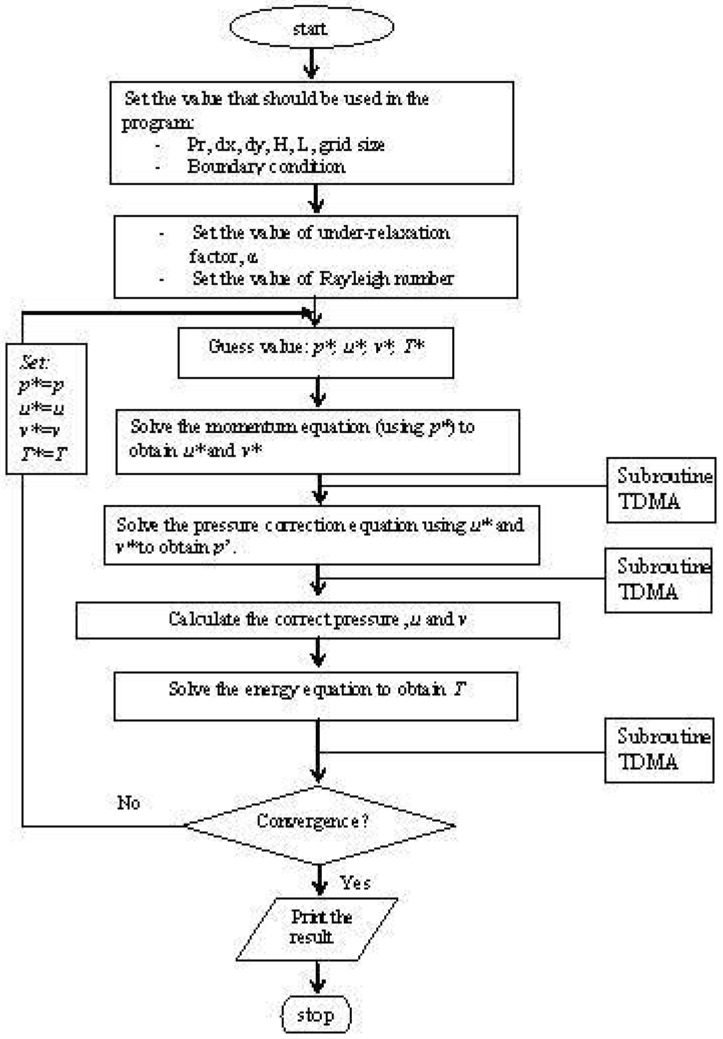

start
Set the value that should be used in the program:
- Pr, dx, dy, H, L, grid size
- Boundary condition
- Set the value of under-relaxation factor, α
- Set the value of Rayleigh number
Guess value: p*, u*, v*, T*
Solve the momentum equation (using p*) to obtain u* and v*
Subroutine TDMA
Solve the pressure correction equation using u* and v* to obtain p'.
Subroutine TDMA
Calculate the correct pressure, u and v
Solve the energy equation to obtain T
Subroutine TDMA
Convergence?
No
Yes
Set: p*=p u*=u v*=v T*=T
Print the result
stop

Figure 2 Flow chart for SIMPLE algorithm.

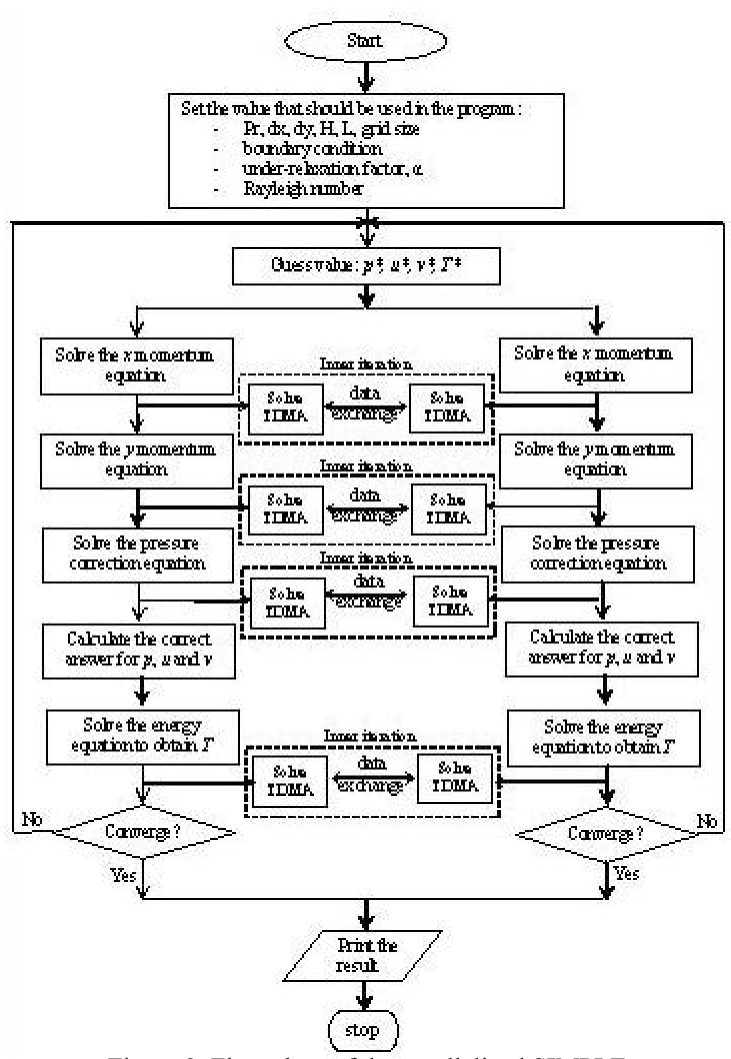

Start
Set the value that should be used in the program :
- Pr, dx, dy, H, L, grid size
- boundary condition
- under-relaxation factor, α
- Rayleigh number
Guess value: p*, u*, v*, T*
Solve the x momentum equation
Inner iteration
Solve TDMA
data exchange
Solve TDMA
Solve the y momentum equation
Solve the pressure correction equation
Calculate the correct answer for p, u and v
Solve the energy equation to obtain T
Converge ?
No
Yes
Print the result
stop

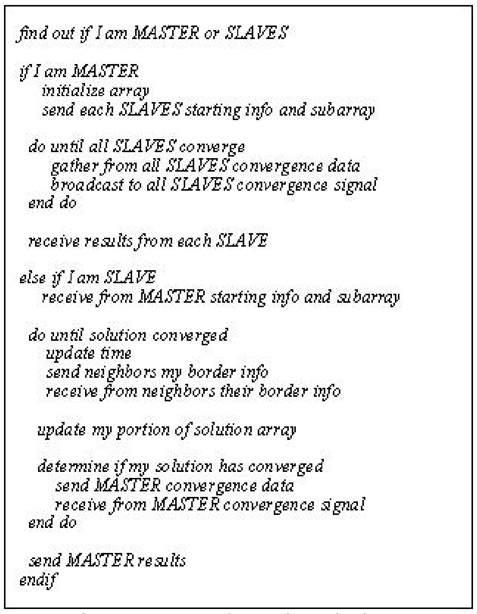
Figure 3 Flow chart of the parallelized SIMPLE with Domain Decomposition Method

4 PARALLEL IMPLEMENTATION

A parallel implementation can provide a further reduction in computing time. Parallel implementation also makes solution possible to problems that would require too much memory to solve on a single processor. During to solve this problem, the parallel implementation is based on message passing (distributed memory systems) using the Parallel Virtual Machine (PVM) software. Portability is ensured because PVM is available on many types of parallel computers.

The implementation uses a layer of subroutines on top of PVM, symbolically denoted by;
- start: start entire parallel application
- stop: stop parallel application
- send: send a message
- receive: receive a message

4.1 Communication Process



```
find out if I am MASTER or SLAVES

if I am MASTER
    initialize array
    send each SLAVES starting info and subarray

  do until all SLAVES converge
      gather from all SLAVES convergence data
      broadcast to all SLAVES convergence signal
  end do

  receive results from each SLAVE

else if I am SLAVE
    receive from MASTER starting info and subarray

  do until solution converged
     update time
     send neighbors my border info
     receive from neighbors their border info

    update my portion of solution array

    determine if my solution has converged
       send MASTER convergence data
       receive from MASTER convergence signal
  end do

  send MASTER results
endif
```

Figure 4 Pseudo code solutions.

Communication process is the most important process in parallel implementation. As described above, the implementation uses a layer of subroutines on top of PVM, denoted by start, stop, send and receive. For the send and receive subroutines, it consists of communication process between a data or function that will be send or receive. According to the pseudo code solution in Figure 4, the communication process occurs between the master and slave during to their sending and receiving the data or function.

4.2 Communication

Basically this finite difference problem is same with the solution of the problem in this project.

From top to bottom of the Figure 5; the one-dimensional vector X, where N=4; the task structure, showing the 4 tasks, each encapsulating a single data value and connected to left and right neighbors via channels; and the structure of a single task, showing its two inports and outports.

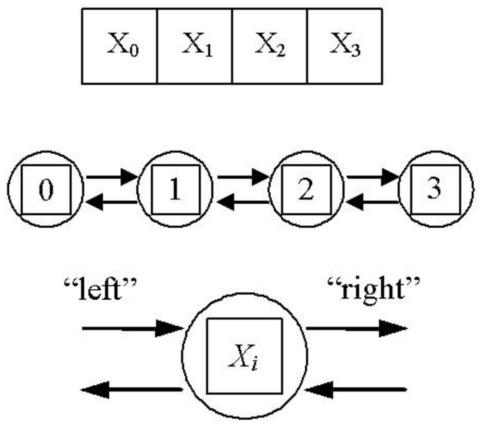


Figure 5 A parallel algorithms for the finite difference problem.

We first consider a one-dimensional finite difference problem, in which we have a vector of size N and must compute, where;

That is, we must repeatedly update each element of X, with no element being updated in step t+1 until its neighbors have been updated in step t. A parallel algorithm for this problem creates N tasks, one for each point in X. The ith task is given the value and is responsible for computing, in T steps, the values.

Hence, at step t, it must obtain the values and from tasks i-1 and i+1. We specify this data transfer by defining channels that link each task with "left" and "right" neighbors, as shown in Figure 5, and requiring that at step t, each task i other than task 0 and task N-1.

i.   sends its data on its left and right outports,
ii.   receives and from its left and right inports, and
iii.   use these values to compute .

Notice that the N tasks can execute independently, with the only constraint on execution order being the synchronization enforced by the receive operations. This synchronization ensures that no data value is updated at step t+1 until the data values in neighboring tasks have been updated at step t. Hence, execution is deterministic.

Figure 6 and 7 below showed the algorithms for the sending and receiving data from master and slaves.

```
C     broadcast data to slaves

      call pvmfinitsend  (PVMDEFAULT, info)
      call pvmfpack (INTEGER4, nproc, 1, 1, info)
      call pvmfpack (INTEGER4, tids, nproc, 1, info)
      call pvmfpack (INTEGER4, n, 1, 1, info)
      call pvmfpack (REAL8, data, n, 1, info)
      msgtype = 1
      call pvmfmcast (nproc, tids, msgtype, info)

C     wait for results from slaves

      msgtype = 2
      do 30 i = 1,nproc
      call pvmfrecv (-1, msgtype, info)
      call pvmfunpack (INTEGER4, who, 1, 1, info)
      call pvmfunpack (REAL8, result(who+1), 1, 1, info)
      if (who.eq.0)
      then
      write (*,1000) result(who+1), who, (nroc-1)
      else
      write (*,1000) result(who+1), who, 2*(who-1)
30    continue
```

Figure 6 Algorithm master to send and receive data to and from slaves.

```
C     receive data from master

      msgtype = 1
      call pvmfrecv  (mtid, msgtype, info)
      call pvmfunpack (INTEGER4, nproc, 1, 1, info)
      call pvmfunpack (INTEGER4, tids, nproc, 1, info)
      call pvmfunpack (INTEGER4, n, 1, 1, info)
      call pvmfunpack (REAL8, data, n, 1, info)

C     determine which slave I'm (0...nproc-1)

      do 5 i = 0,nproc
      if (tids(i).eq.mytid) me = i
5     continue

C     do calculation with the data

      result = work (me, n, data, tids, nproc)

C     send the result to the master

      call pvmfinitsend (PVMDEFAULT, info)
      call pvmfpack (INTEGER4, me, 1, 1, info)
      call pvmfpack (REAL8, result, 1, 1, info)
      msgtype = 2
      call pvmfsend (mtid, msgtype, info)
```

Figure 7 Algorithm slaves to receive and send data from and to master.

4.3 Physical Model

The above algorithm has been tested for the problem of natural convection that occurred in a square cavity with specified boundary conditions. The flow in a square cavity considered here is that has a hot left vertical wall temperature and cold right vertical wall. The upper and lower walls are adiabatic. The fluid used is air with a Prandtl number of 0.71. The aspect ratio L/H is 1.

The flow is described by the Navier-Stokes equations under the Boussinesq approximation that will discuss later. The summary of the boundary condition that was chosen are as below:

i. Adiabatic at both upper and lower wall.
ii. For the vertical wall, the boundary condition is isothermal with hot at left vertical wall and cold at right vertical wall.
iii. Velocities at the boundary are zero.

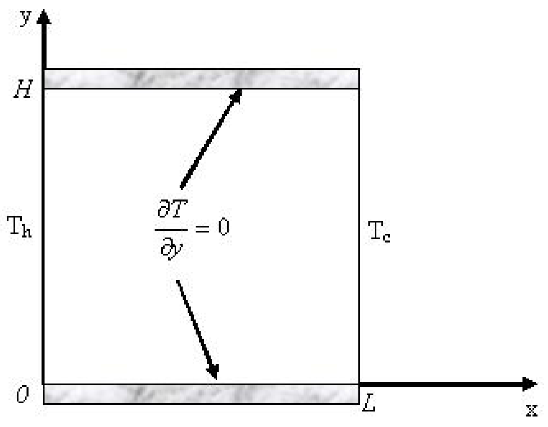


Figure 8 Model of square cavity

5 DISCUSSION

5.1 Validation of the Results

Tables 1 to 3 show the comparison between the results from the present simulation and the literature results obtained by Davis (1983). The results of Davis are the standards against which all other codes are evaluated. Maximum horizontal velocity on the vertical midplane of the cavity, Umax, maximum vertical velocity on the horizontal midplane of the cavity, Vmax, and an average of Nusselt number were compared at Rayleigh numbers of 103, 104, 105 and 106. The comparison had been done between the benchmarking results obtained by Davis which in serial processor and the present study that are simulation using serial processor and parallel processor or parallel computer.

From the tables, it showed that all these results are in excellent agreement with the benchmark results of Davis. Percentage error for the three methods of solution was below than 3% compare with benchmark result. Besides that, the result that was showed in the forms of contour maps of non-dimensional temperature and velocities was also compared with the results that obtained by Davis.

Figures 9 to 20 showed the contour maps of non-dimensional temperature and velocities. The thermal and flow fields also agree very well with those reported by Davis (1983).

Table 1 Comparison of the numerical result of present study for Umax

Ra	10^3	10^4	10^5	10^6
G. de Vahl Davis	3.649	16.193	34.620	64.593
Present study:				
i) Serial processor	3.652	16.163	34.871	65.812
% error	0.082 %	0.185 %	0.725 %	1.880 %
ii) Parallel processor	3.592	16.376	34.852	65.847
% error	1.560 %	1.131 %	0.670 %	1.941 %

Table 2 Comparison of the numerical result of present study for Vmax

Ra	10^3	10^4	10^5	10^6
G. de Vahl Davis	3.697	19.167	68.590	216.360
Present study:				
i) Serial processing	3.704	19.675	69.482	220.641
% error	0.189 %	2.650 %	1.300 %	1.978 %
ii) Parallel processing	3.715	19.642	69.680	221.282
% error	0.487 %	2.478 %	1.589 %	2.275 %

Table 3 Comparison of the numerical result of present study for $\overline{Nu}$

Ra	10^3	10^4	10^5	10^6
G. de Vahl Davis	1.118	2.243	4.519	8.800
Present study:				
i) Serial processing	1.120	2.282	4.583	8.983
%error	0.23%	1.74%	1.42%	2.08%
ii) Parallel processing	1.123	2.272	4.594	9.008
%error	0.47%	1.31%	1.67%	2.36%

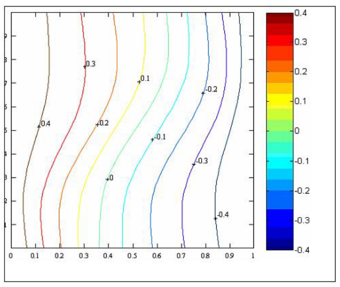


Figure 9 Contour maps of temperature, Ra = 103

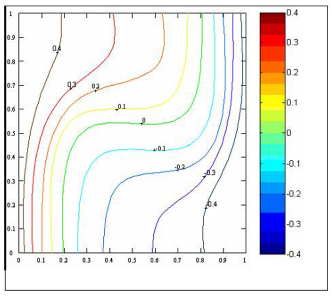


Figure 10 Contour maps of temperature, Ra = 104

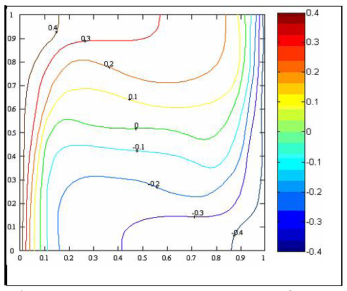


Figure 11 Contour maps of temperature, Ra = 105

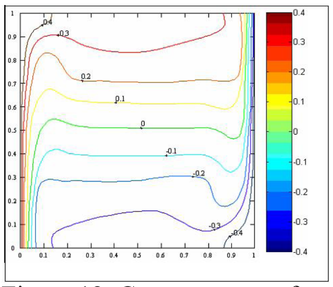


Figure 12 Contour maps of temperature, Ra = 106

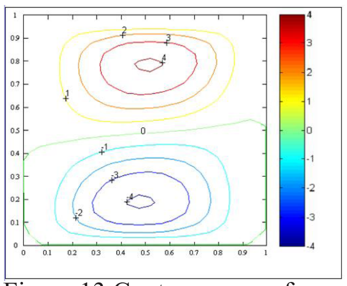


Figure 13 Contour maps of temperature, Ra = 103

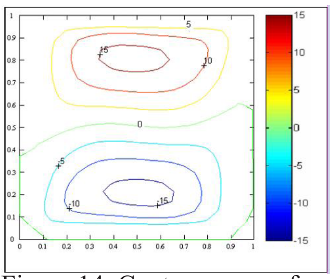


Figure 14 Contour maps of temperature, Ra = 106

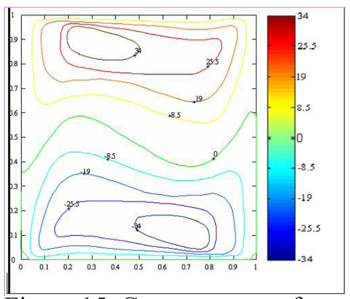


Figure 15 Contour maps of horizontal velocity u, Ra = 105

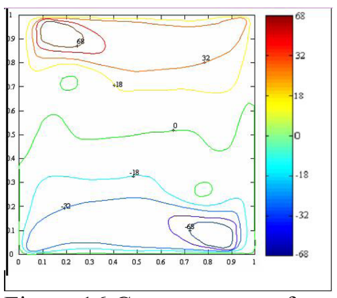


Figure 16 Contour maps of horizontal velocity u, Ra = 106

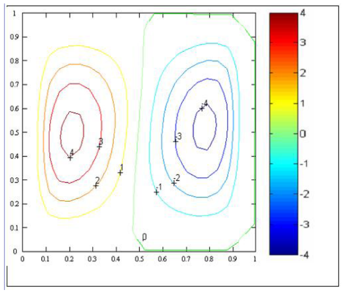


Figure 17 Contour maps of vertical velocity v, Ra = 103

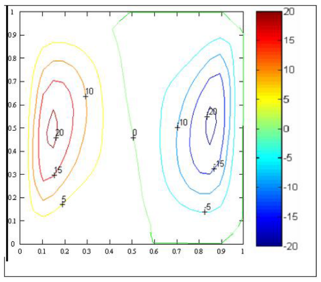


Figure 18 Contour maps of vertical velocity v, Ra = 10

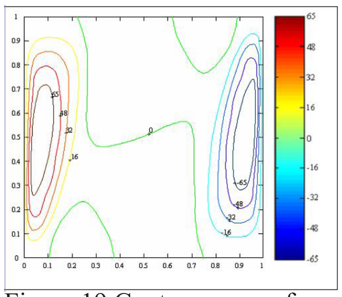


Figure 19 Contour maps of vertical velocity v, Ra = 105

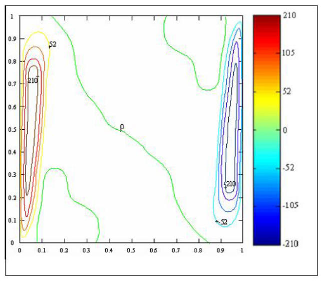


Figure 20 Contour maps of vertical velocity v, Ra = 106

6 CONCLUSION

A parallel algorithm has been developed to simulate an incompressible flow for the problem of natural convection that occurred in a square cavity with specified boundary conditions. The simulations of the incompressible flow using SIMPLE method on parallel computer are agreement with the benchmark result. Thus, the simulation is successful. Percentage errors for the two computational solutions which are simulation by serial and parallel computer are below than 3% compare with benchmark result by Davis. Besides that, the contour maps of temperature T, contour maps of horizontal velocity u and contour maps of vertical velocity v also agree very well with those reported in the literature. Therefore it has proved that clustering personal computers together can provide adequate computing power for large engineering problems.

REFERENCES

[1] Date A. W (1985). “Numerical Prediction of Natural Convection Heat Transfer in Horizontal Annulus.” Int. J. Heat Mass Trasfer.

[2] Davis G. de Vahl (1983). “Natural convection of air in a square cavity: a benchmark numerical solution”. Int. Journal Numerical Mech. Fluid (3): 249-264.

[3] Dongarra, J. & Eijkhout, U. (2000). “Numerical linear algebra algorithms and software.” Journal of Computational and Applied mathematics. 123 (2):489-514.

[4] Geist, A. et al. (1994). “PVM: Parallel Virtual Machine. A Users’ Guide and Tutorial for Networked Parallel Computing”. Massachusetts: The MIT Press.

[5] Melaaen M. C (1993). “Nonstaggered Calculation of Laminar and Turbulent Flows using Nonorthogonal Coordinates.” Int. J. Num. Heat Transfer: 375-392.

[6] Patankar S. V (1980). “Numerical Heat Transfer and Fluid Flow.” McGraw-Hill Inc, New York.

[7] Patankar S.V and Spalding D. B (1972). “A Calculation Procedure for Heat, Mass and Momentum Transfer in 3-Dimensional Parabolic Flows.” Int. J. Heat Mass Trasfer.

[8] Spalding D. B (1972). “A Novel Finite Difference Formulation for Differential Expressions Involving Both First and Second Derivatives.” Int. J. Num. Methods Eng. (3): 551-559.

[9] Versteeg H. K and Malalasekera M. (1995). “An Introduction to Computational Fluid Dynamics.” Pearson, Prentice Hall.