İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

PARALLEL SOLUTION OF UNSTEADY, INCOMPRESSIBLE THREE-DIMENSIONAL NAVIER-STOKES EQUATIONS WITH A NEW IMPLICIT METHOD

Ph.D. Thesis by

Vildan ÜSTOĞLU ÜNAL

Department : **Astronautical Engineering**

Programme: **Astronautical Engineering**

JANUARY 2003

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

PARALLEL SOLUTION OF UNSTEADY, INCOMPRESSIBLE THREE-DIMENSIONAL NAVIER-STOKES EQUATIONS WITH A NEW IMPLICIT METHOD

Ph.D. Thesis by
Vildan ÜSTOĞLU ÜNAL

(511972105)

Date of submission : 28 June 2002

Date of defence examination: 7 January 2003

Supervisor (Chairman): Prof. Dr. Ülgen GÜLÇAT

Members of the Examining Committee Prof.Dr. A. Rüstem ASLAN (İ.T.Ü.)

Prof.Dr. M. Fevzi ÜNAL (İ.T.Ü.)

Prof.Dr. Hasan HEPERKAN (Y.T.Ü.)

Assoc. Prof.Dr. Haluk ÖRS (B.Ü.)

JANUARY 2003

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

PARALLEL SOLUTION OF UNSTEADY, INCOMPRESSIBLE THREE-DIMENSIONAL NAVIER-STOKES EQUATIONS WITH A NEW IMPLICIT METHOD

Ph.D. Thesis by
Vildan ÜSTOĞLU ÜNAL

(511972105)

Date of submission : 28 June 2002

Date of defence examination: 7 January 2003

Supervisor (Chairman): Prof. Dr. Ülgen GÜLÇAT

Members of the Examining Committee Prof.Dr. A. Rüstem ASLAN (İ.T.Ü.)

Prof.Dr. M. Fevzi ÜNAL (İ.T.Ü.)

Prof.Dr. Hasan HEPERKAN (Y.T.Ü.)

Assoc. Prof.Dr. Haluk ÖRS (B.Ü.)

JANUARY 2003

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ZAMANA BAĞLI, SIKIŞTIRILAMAZ, ÜÇ BOYUTLU NAVİER-STOKES DENKLEMLERİNİN YENİ BİR KAPALI METODLA PARALEL ÇÖZÜMÜ

DOKTORA TEZİ

Vildan ÜSTOĞLU ÜNAL
(511972105)

Tezin Enstitüye Verildiği Tarih : 28 Haziran 2002
Tezin Savunulduğu Tarih : 7 Ocak 2003

Tez Danışmanı :	**Prof. Dr. Ülgen GÜLÇAT**
Diğer Jüri Üyeleri	**Prof.Dr. A. Rüstem ASLAN (İ.T.Ü.)**
	Prof.Dr. M. Fevzi ÜNAL (İ.T.Ü.)
	Prof.Dr. Hasan HEPERKAN (Y.T.Ü.)
	Doç. Dr. Haluk ÖRS (B.Ü.)

OCAK 2003

ACKNOWLEDGEMENTS

After the valuable education in aerodynamics and computational fluid dynamics at I.T.U. Faculty of Aeronautics and Astronautics, this work has been created with the guidance of my supervisor Prof. Dr. Ülgen Gülçat, who motivated me and enlarged my knowledge of CFD world during the coarse of this study. I also would like to thank him and Prof. Dr. A. Rüstem Aslan for granting permission to use and modify the new implicit algorithm, which has been developed earlier by them.

I would like to thank to the members of my Ph.D. Examining Committee, Prof. Dr. M. Fevzi Ünal, Prof. Dr. A. Rüstem Aslan and Assoc. Prof. Dr. Fırat Oğuz Edis, who continuously guided, followed and contributed to my researches. Assoc. Prof. Dr. Aydın Mısırlıoğlu never hesitated to help despite his workload. With the guidance of them and my supervisor, I overcame my problems with finite element method, enlarged my knowledge of aerodynamics, computational fluid dynamics and parallel programming.

Without the moral support of my friends at I.T.U. and Yeditepe University, it would be hard to go on the studies.

I would like to thank all the members of my great family for their support during this prolonged study. My father in law, who is an academician, helped me during my doctoral studies with his experiences and advices.

I owe much more than a thank to my father, my mother and my sisters. My success is theirs. They were always with me with their love and understanding and I will never forget their sacrifices for me.

Special thanks to my husband Burak. Without his moral support, patience and encouragement, it would most certainly not have been possible to overcome the stress of this work.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

A : Sparse coefficient matrix

φ : Solution vector

B : Source vector

$\mathbf{F}_{viscous}$: Viscous force

$\mathbf{F}_{pressure}$: Pressure force

$\mathbf{F}_{body}$: Body force

ρ : Density

$\mathbf{u}=(u, v, w)$: Velocity vector

t : Time

p : Pressure

μ : Dynamic viscosity

v : Kinematic viscosity

U_o : Characteristic velocity

L : Characteristic length

$'$: Non-dimensional

Re : Reynolds number

n : Old time-step level

$n+\frac{1}{2}$, n+1 : Half and full time-step levels

$\mathbf{u}^*$: Intermediate-time step velocity

Δt : Time-step

$d\Omega$: Volume element

h_α : Boundary data

$d\Gamma$: Boundary of volume element

α : Cartesian coordinates x, y, z

i, j, k : Element node numbers

$\mathbf{D}$: Advection matrix

$\mathbf{A}$: Stiffness matrix

$\mathbf{C}_\alpha$: Coefficient matrix for pressure

$\mathbf{B}_\alpha$: Matrix arising due to the boundary conditions

$\mathbf{M}_{ij}$: Mass matrix

e : Element value

ϕ : Auxiliary potential function

L : Lower triangular matrix

U : Upper triangular matrix

Ω_i : i th sub-domain

S_j : j th interface

o : Initial interface iteration level

n : Interface iteration level

y_i : Unknowns at i th sub-domain

$\varepsilon_{\mathbf{u}}$: Interface iteration criteria for velocity field

ε_ϕ : Interface iteration criteria for auxiliary potential

ε_p : Interface iteration criteria for pressure

k : Interface iteration converged value

$\overline{\mathbf{A}}$: Coefficient matrix

μ^k : converged Neumann condition at interface

w^n : not-converged Neumann condition at interface

ZAMANA BAĞLI, SIKIŞTIRILAMAZ, ÜÇ BOYUTLU NAVİER-STOKES DENKLEMLERİNİN YENİ BİR KAPALI METODLA PARALEL ÇÖZÜMÜ

ÖZET

Bilgisayar uygulamalı metodlar, sayısal tekniklerin gelişimi ve bilgisayar performansındaki artışla, endüstride ekonomik bir dizayn ve analiz aracı haline gelmiştir. Akışı, türbülans veya karmaşık akışlar gibi, hassas olarak simüle edebilmek için bilgisayar kapasitesinin artmasına duyulan talep halen mevcuttur. Günümüz bilgisayarlarının sınırlı gücü, ekonomik analizlerin yapılması açısından, bilgisayar uygulamalı algoritmaların gelişmesini gerektirmektedir. Bu talepler göz önünde bulundurulduğunda, paralel hesaplama, günümüzde akışın etkin bir şekilde çözümlenmesi ve hassas sunumunun elde edilmesinde önemli bir rol almaktadır.

Bu çalışmanın amacı, laminer ve türbülans akışların modellenmesinde temel araç olan zamana bağlı, sıkıştırılamaz, üç boyutlu Navier-Stokes denklemlerinin sayısal çözümü için yeni bir kapalı metod kullanarak, etkin bir paralel algoritma kurmak ve uygulamaktır. İncelenen problemlerin büyüklüğü ve karmaşıklığı, paralel hesaplamanın etkin kullanımını gerektirmektedir. Paralel programlama öncelikle hesaplamaları hızlandırmak için kullanılır, bir paralel algoritma mümkün olan en iyi sıralı çözümden bile oldukça hızlı olabilir, bu şekilde hızlı çözümlere özellikle ihtiyaç duyulmaktadır. Paralel hesaplama uygulamaları gün geçtikçe artmaktadır. Bu uygulama alanlarında, günümüzde veya gelecekte hiçbir klasik bilgisayarın sağlayamayacağı kadar yüksek hesaplama hızına ihtiyaç duyulmaktadır. Paralel hesaplama tekniği, günümüzde bu hesaplamaları mümkün kılabilecek tek yaklaşım olarak bilinmektedir. Bu çalışma, paralel algoritmaların gelişimine katkıda bulunmayı amaçlamıştır.

Hafıza gereksinimi ve hesaplama zamanı dikkate alınması gereken önemli konular olduğundan, denklemlerin ayrıklaştırılmasında kullanılacak şema yüksek derecede hassas, hızlı ve verimli olmalıdır. Genellikle çok fazla nokta kullanılarak çözümlenebilen karmaşık akışlarda, açık şemalar kararlılık şartlarından dolayı atılacak zaman adımı büyüklüğüne önemli kısıtlamalar getirmektedir. Bu nedenle, bu tarz akışların incelenmesinde kapalı çözücüler tercih edilmektedir. Kapalı algoritmalar, daha büyük zaman adımları kullanabilir. Çözüme ulaşmak için atılacak zaman adımlarındaki azalmadan dolayı, bloklar arasındaki alışverişin de azalması sayesinde, bir kapalı şema, paralel hesaplamalar söz konusu olduğunda, açık şemaya göre daha avantajlı hale gelebilir.

Bu çalışmada ilk adım, bazı algoritmaların hassasiyet ve hesaplama zamanı açısından, kapalı şema kullanılan programı sıralı olarak koşturarak test edilmesi ile atıldı. Paralel kapalı şema kullanılacak olan programlamaya geçmeden önce, paralel işlemciliğe en uygun olan algoritma seçildi. Programın sıralı olarak koşturulmasıyla

test edilen dört yöntem arasında, direkt çözümle yardımcı potansiyel fonksiyonunu elde eden formülasyon şekli, gerek hesaplama zamanı gerekse paralel programlama uygulama kolaylığı açısından, en verimli metod olarak gözlenmiştir. Kapalı şema kullanan program, basınç alanına ulaşmakta kullanılacak olan yardımcı potansiyel fonksiyonunu direkt çözümle elde edecek şekilde yenilenmiştir.

Bu tez yeni bir kapalı metodu araştırmakta ve temel denklemlerin kapalı çözümünde kullanılacak olan yeni bir paralel algoritmayı geliştirmektedir. Momentum denkleminin zaman ayrıklaştırmasında yenilenmiş kesirli zaman adımı metodu kullanılırken, uzayda ayrıklaştırma Galerkin Sonlu Elemanlar Metodu ile brik elemanlar kullanılarak yapılmaktadır. Kullanılan şema zamanda ve uzayda ikinci mertebedendir. Bu da akış alanının, atılacak zaman adımı büyüklüğüne bir sınırlama getirmeden az sayıda nokta kullanılarak çözümlenmesine izin vermekte, böylelikle kapalı şemada kullanılan matrislerin boyutu da küçülmektedir. Düzgün ve düzgün olmayan ağ yapıları kullanılmıştır. Düğüm numaralandırması, özellikle en az yarı bant genişliğini verecek ve böylelikle hafıza açısından kazanç sağlayacak ve hesaplama zamanını azaltacak şekilde düzenlenmiştir.

Her zaman adımında momentum denklemi, yarı zaman adımı hız alanı elde etmek için paralel olarak çözülmektedir. Yarı zaman adımı hız alanı kullanılarak, Poisson denklemini sağlayan yardımcı potansiyel fonksiyonu paralel olarak hesaplanmakta ve basınç alanına ulaşılmaktadır. Paralel performans değerlendirmesi için, yardımcı potansiyel fonksiyonu yerine direkt basıncı hesaplama yolu da test edilmiş, fakat yardımcı potansiyel fonksiyonunu kullanan Poisson denkleminin paralel çözümünün direkt basınç formülasyonu kullanan paralel çözüme göre daha az hesaplama zamanı aldığı gözlenmiştir.

Bölgelere ayırma tekniğine dayanan paralel hesaplamalar, büyük ölçekli veya karmaşık akış problemlerinin çözümünü mümkün kılmaktadır. Matris denklemlerimizin paralel kapalı çözümünde, yenilenmiş Bölgelere Ayırma Metodu, momentum ve Poisson denklemlerinin örtüşmeyen fakat eşleşen alt bölgeler kullanılarak paralel çözümü için, etkin hale getirilmiştir. Çözüm alanı bir çok alt bölgelere, *köleler*, ayrılır ve iş istasyonlarına dağıtılır. Köle işlemciler arakesit verilerini başlangıç ve sonuçlandırma işlemlerinden sorumlu *ana* işlemci ile değiştirirler. Hesaplamalar ilerledikçe, köleler sınır bilgilerini de ana işlemci ile periyodik olarak değiştirirler. İşlemler sonrasında global çözüm, işlemcilerdeki çözümlerin bir araya getirilmesi ile ortaya konur. Arakesitlerde doğru Neuman şartları elde edilinceye kadar arakesit iterasyon döngüsü devam eder. Bu tezde gerek yarı zaman adımı hız alanı gerekse yardımcı potansiyel fonksiyonu için en iyi arakesit iterasyon kriterleri çok sayıda denemeler sonucu bulunmuş, ve kararlı ve hassas çözüm için gereken en iyi kriterler sunulmuştur. Bulunan doğru arakesit şartlarıyla beraber her bölgede denklemler birbirinden bağımsız olarak çözülür ve sonunda birleştirilerek global çözüm tamamlanmış olur. İç iterasyon döngüsü ve dış zaman adımı gelişmelerini kapsayan hesaplamalar, her işlemcide ana işlemciyle haberleşilerek paralel bir şekilde çıkarılmalıdır.

İlk paralel çalışmalar, birbirleriyle bir 100 Mbps TCP/IP ağ yardımı ile bağlı iş istasyonlarının (DEC Alpha XL266) gerektiğinde PVM 3.3 (Parallel Virtual Machine) ile haberleşmesi ile Linux işletim sisteminde yapılmış, daha sonra ise bu

çalışmalar, sekiz işletimcinin bulunduğu yeni bir paralel Unix işletim sisteminde (SGI O3000) devam ettirilmiştir.

Yeni kapalı paralel şema uygulaması, kapağı çekilen küp oyuk içerisindeki, Reynolds sayısı 400 ve 1000 alınan akım problemi için kalibre edilmiştir. Hesaplamalar, toplam hesap noktası (21x11x21), (25x13x25), (49x25x49) ve (55x28x55) olan, 2, 4 ve 6 bölgeli ayrımlarda devam etmiştir. 0.1 gibi büyük bir zaman adımı ve en küçük 0.014 ağ büyüklüğü kullanılarak, 25 boyutsuz zamanda yakınsama sağlanmıştır. Çözümler, referanslarla karşılaştırılmıştır. Referanslardakilere göre çok az sayıda çözüm noktası kullanılmasına rağmen, iyi bir uygunluk sağlanmıştır. Özellikle sonuçların referans değerlerine yakınlaşmasının artmasında, oyuk duvarlarına yakın bölgelerde sıkı ağ kullanılması etkili olmaktadır. Kapağı çekilen küp oyuk içerisindeki ana ve ikincil akışlar ve Reynolds sayısı etkileri, değişik kesitler için sunulmuştur. Akışın simetrik karakteri, tüm oyuk geometrisi ele alınarak yapılan sayısal çözümle gösterilmiş ve bu tezde hesaplamalar tümüyle simetrik yarıyı kapsamıştır.

DEC Alpha XL266 iş istasyonlarında, eşit sayıda çözüm noktası kullanılan 2, 4 ve 6 bölgeli ayrıklaştırmalar üzerinde elde edilen ilk paralel sonuçlar, bu iş istasyonlarında yük dengelemesi yapılmasının gerekliliğini göstermiştir. Burada hızlanma yeterli derecede olmamıştır. İki ayrı paralel çözüm şeması uygulanmıştır. Hızlanmayı arttırmak için yük dengelemesi yapılmış ve ideal hızlanma elde edilmiştir. Yük dengelemesi, ana işlemcinin yavaşlığından dolayı uğranılan hızlanma kaybını önlemek üzere, neredeyse tüm paralel hesaplamaların köleler üzerinde yapılması esasına dayandırılmıştır. Kapağı çekilen küp oyuk problemi için paralel çözüm, 2 bölge kullanılarak 4105, 4 bölge ile 1719 ve 6 bölge ile 1579 saniyede (toplam hesaplama zamanı), DEC Alpha XL266 iş istasyonları üzerinde, elde edilmiştir.

DEC Alpha iş istasyonlarında hızlanmayı arttırmak için gerekli olan bu paralelleştirme yapısı, SGI O3000 sisteminde ilk başta uygulanan paralel yapıya göre daha olumsuz sonuç vermiştir. Paralel uygulamada kullanılan paralel işlemciler eğer birbirine nitelik, hız, haberleşme hızı açısından denk ise, iyi bir paralel performans için, problemi çözmek için gereken toplam hesaplamalar eşit olarak işlemciler arasında paylaştırılır. Tablo 7.6'da da görülebileceği gibi, bütün arakesit datasını işleyecek olan ana işlemcinin, kölelerden daha hızlı olması, paralel performansı arttırmaktadır.

Bölüm 9, Kısım 9.3' de açıklanan bölgelere ayrıklaştırma denklemlerinde yapılan son yeniliklerle, özellikle çok fazla miktarda olan yardımcı potansiyel fonksiyonu arakesit iterasyonları yaklaşık yüzde elli oranında azaltılmıştır. Arakesit Neumann sınır şartları, her zaman adımı başında, sıfır yerine, bir önceki zaman adımı arakesit iterasyonları sonucu yakınsamış değere eşitlenmiştir. Bölgelere Ayırma Metodunun başlangıç bölümünde, her alt bölge, bir önceki zaman adımında yakınsamış değere eşitlenmiş Neumann şartı ile beraber Poisson denklemini çözer. Bölgelere Ayırma Metodu, bu düşünceye göre düzenlenmiştir ve sonunda arakesit hesaplamaları daha kolay yakınsamış ve iterasyon sayısı etkin bir şekilde azaltılmıştır. Arakesit sayısı arttıkça iterasyon sayısı da arttığından, iterasyon sayısının düşürülmesi, aynı zamanda baş ve köle işlemciler tarafından yapılan paralel hesaplamaların da

azalmasını sağlamıştır. Böylelikle hızlanma artmıştır. Üstelik hesaplama zamanı azalmış ve süper-lineer hızlanma başarılmıştır.

Bu son yeniliklerle, SGI Origin 3000 sisteminde, 2 bölgeli paralel çözüm zamanına (toplam hesap zamanı) göre hesaplanmış hızlanma değeri, 2.30'dan 2.71'e 4 bölgeli ayrıklaşmayla ve 2.40'dan 3.12'ye 6 bölgeli ayrıklaşmayla yükselmiştir. Hızlanma, kölelerin CPU zamanı kullanılarak hesaplandığında, dört bölgede 3.31'e ve 6 bölgeli hesaplamalarda 5.69'a kadar çıkmıştır.

PARALLEL SOLUTION OF UNSTEADY, INCOMPRESSIBLE THREE-DIMENSIONAL NAVIER-STOKES EQUATIONS WITH A NEW IMPLICIT METHOD

SUMMARY

Computational methods became a cost-effective design and analysis tool for the industry with the increasing performance of computers and developments in numerical techniques,. Demand for increased computational effort for accurate simulation of the flow, such as turbulence or complex flows, still exists. The limited computational power of the day requires the development of computational algorithms to perform cost effective analysis. Considering these demands, parallel computation became an important task in recent years in order to get efficient solutions and accurate representation of the flow.

The objective of this study is to construct and apply an efficient parallel algorithm using a new implicit method for the solution of unsteady, incompressible three-dimensional Navier-Stokes equations, which are the fundamental tool for modelling laminar and turbulent flows. The size and complexity of the problems considered necessitates the utilization of parallel computing. Parallel programming is used primarily to speed up computations. A parallel algorithm can be significantly faster than the best possible sequential solution. Such fast solutions are truly needed. There is a growing number of applications of parallel computing. These fields require computing speeds that can not be supplied by any current or future conventional computer. Parallel computation is the only approach known today that would make these computations feasible. This study aims to contribute towards the development of parallel algorithms.

The scheme in discretizing the equations must be of high-order accuracy, fast and efficient, since memory requirements and computing times are important subjects to be considered. Because of the stability requirements, explicit schemes impose severe restrictions on the time step size for complex flows, which are generally resolved with sufficiently fine grids. Therefore, the implicit flow solvers are preferred in the study of such flows. An implicit algorithm can use larger time steps. Because of the reduced number of time-steps to reach the solution, an implicit solver may outperform an explicit one in parallel computing by reducing the total amount of inter-block communications.

Thus, in this study, the first step is realized by testing some algorithms by the implicit sequential runs, considering the accuracy and computational time. The most suitable one for parallel processing is chosen before starting the parallel programming. Among the four methods tested by the sequential runs, the direct auxiliary potential formulation has been observed to be more efficient in terms of

computational time and easier programming. The implicit computational code has been modified in such a way that the solution is obtained by the direct auxiliary potential function formulation for the pressure.

This thesis investigates a new implicit method and develops a new parallel algorithm for the implicit solution of the governing equations. The space is discretized with brick elements by the Galerkin Finite Element Method, while modified version of the two-step fractional method is used in time discretization of the momentum equation. The scheme is second-order accurate in both time and space. This allows us to resolve the flow field with less number of points while taking large time steps so that the size of the matrix to be inverted in the implicit method becomes small. Both uniform and nonuniform grids are used. The node numbering is specially arranged to obtain the minimum half-bandwidth, to save from computer storage and to have an advantage in terms of computational time.

At each time step, the momentum equation is solved by parallel computation to get the half time-step velocity field. Auxiliary scalar potential which satisfies Poisson's equation is obtained using the half time-step velocity field and the pressure at each time level is obtained. Direct pressure calculation instead of using auxiliary potential function is also tested for the parallel performance, but it is observed that the parallel solution of Poisson's Equation with auxiliary potential function formulation takes less computational time than that with the direct pressure formulation.

Domain decomposition-based parallel computing makes it possible to solve large size or complex problems. For the parallel implicit solution of the matrix equations, modified versions of the Domain Decomposition Method is utilized for parallel solution of both the momentum and Poisson equations using non-overlapping matching sub-domains. The computational domain is divided into several sub-domains, or *slaves*, and distributed among the workstations for concurrent processing. The slave processors exchange the interface data with the *master* processor, which is responsible for pre- and post-processing. Slaves periodically exchange boundary information with the master, as the computation progresses. The global solution is then put together during post-processing. Interface iteration cycle goes on until the correct interface Neumann boundary condition is found. In this thesis the best interface iteration convergence criteria for both half-time level velocity field and the auxiliary potential function is found with a lot of computational experiments, and the best ones for the stable and accurate solution is presented. The global solution is then assembled by solving the equations for each block separately together with the correct interface conditions. The computations involving an inner iterative cycle and outer time step advancements have to be performed in a parallel manner on each processor communicating with the master processor.

The first results are obtained on a cluster of DEC Alpha XL266 workstations running Linux operating system, interconnected with a 100 Mbps TCP/IP network. Public version of the PVM 3.3 is used as the communication library. But the latest studies are developed on SGI Origin 3000 with 8 processor running Unix operating system.

The new implicit parallel implementation is calibrated for the lid-driven cubic cavity problem with Reynolds number of 400 and 1000. Computations are carried on 2, 4 and 6-partitioning domains with total number of 21x11x21, 25x13x25, 49x25x49 and

55x28x55 nodes. The steady state is reached in 25 dimensionless time, with a large step-size of 0.1 and with a minimum grid size of 0.014. The results are compared with the benchmark solutions. They give well agreement although coarser grids are used compared to those used at the benchmark computations. The agreement with the benchmark results is increased with fine grids, especially with fine stretched cells along the walls of the cavity. The primary and secondary flows inside the lid-driven cubic cavity and the effect of Reynolds number are presented for different cross-sections. Full geometry solution of the lid-driven cubic cavity proved that the flow inside the lid-driven cavity is symmetric so that the computation included only the symmetric part.

The first parallelization results on 2, 4 and 6-domain partitioning, having equal number of grid points, show that the parallel computation on DEC Alpha XL266 workstations necessitates the load balancing. In this case, the speed-up is not good enough. Two more paralellization schemes are tested. Load balancing is applied to increase the speed-up on the cluster of workstations and an ideal value of speed-up is obtained. The load balance is done in such a way that the slave processors handle almost all of the domain decomposition calculations so that the decrease in parallel efficiency, caused by the slow master processor, can be prevented. Parallel computation for lid-driven cubic cavity problem takes 4105 seconds with 2-domain, 1719 seconds with 4-domain and 1579 seconds (elapsed time) with 6-domain partitionings, on DEC Alpha XL266 workstations.

The parallelization structure applied to increase the speed-up on DEC Alpha XL266 workstations gives worse results on SGI O3000 system, compared to the parallel structure applied at the beginning of the thesis. If all the processors used in the parallel computations have the same quality, speed and communication speed, all of the parallel work to solve the interested problem should be distributed equally to each processor. As it can be seen in Table 7.6, in parallel computation the speed-up increases efficiently if the speed of master processor, for pre- and post processing, is higher than the speed of slave processors.

With the last modification of the domain decomposition equations explained in Chapter 9 Section 9.3, the number of interface iterations, especially the high number of auxiliary potential iterations, is decreased by approximately 50 percent. The interface Neumann boundary conditions at the beginning of each time level are set not equal to zero, but to the converged value which is found out at the end of the interface iterations at the previous time level. At the initialization part of Domain Decomposition Method, each sub-domain solves Poisson equation together with the Neumann condition set equal to the converged value which is obtained at the end of the old time level interface iterations. The domain decomposition equations are modified with this idea, and finally, interface computations are seen to converge more easily, the number of iterations being reduced efficiently. Since the iterations increase with the increasing number of interface, the reduction of iterations also reduce the parallel calculations done by the master and the slave processors. Consequently, the speed-up increases. Moreover, the computational time is decreased and super-linear speed-up is achieved.

With this latest modification, the speed-up based on the 2-domain computation time (elapsed-time) is increased from 2.30 to 2.71 on 4-domain partitioning, and from

2.40 to 3.12 on 6-domain partitioning, on SGI Origin 3000 system. Speed-up, calculated using slave CPU time, is 3.31 on 4-domain and rises up to 5.69 on 6-domain partitioning computations.

1. INTRODUCTION

Although Computational Fluid Dynamics (CFD) has seen big growth in the past three decades and has been applied to a wide variety of phenomena, ranging from thunderstorms to blood circulation, it is still in development. This is mostly due to the complex mathematical structure of the Euler and Navier-Stokes equations that are used to describe fluid motion.

The most successful numerical schemes developed for CFD in the past have been those based upon the flow physics. While this success is seen in one-dimensional flows, there are still problems in higher dimensions. Not only do we seek a more accurate method of discretizing the equations, but issues of code robustness and rate of convergence are also important areas that need improvement.

Starting from mid 1960's, computational techniques began to have a significant effect on aerodynamic analysis and design. Today, with the increasing performance of computers and developments in numerical techniques, computational methods became a cost-effective design and analysis tool for the industry. Demand for increased computational effort for accurate simulation of the flow, such as turbulence or complex flows, still exists and the limited computational power of the day requires the development of computational algorithms to perform cost effective analysis. Considering these demands, parallel computation became an important task in recent years in order to get the efficient solutions and accurate representation of the flow.

Navier-Stokes equations are the fundamental tool for modelling the laminar and turbulent character of the flow. The equations are a system of second-order partial differential equations of the evolution type and describe general fluid motion; particularly the flow of viscous fluids, where the effects of viscosity, thermal conduction, and mass diffusion are important. The governing equations are obtained through conservation of momentum. The stress and rate of strain components are

defined by Stoke's hypotheses; the viscous forces are expressed explicitly in terms of the appropriate flow-field variables. The resulting equations are called the Navier-Stokes equations, the most pivotal equations in all of theoretical fluid dynamics. The convection terms in the Navier-Stokes equations are non-linear, and therefore, the superposition principle and many classical analytical techniques can not be used. Numerical methods must in general be used to solve this system of equations. So the issues of accuracy, robustness and convergence time of the numerical methods considered for the solution of this system are very important for the CFD world and so very important for real-life applications, [1-4].

Incompressible flow modeling is an important area of general industrial interest. Many industrial processes, devices and environmental problems involve incompressible flow of viscous fluids. If this system can be well understood and solved accurately and efficiently, the practical problems can be cleared out, such as nuclear reactor insulation, ventilation of rooms, solar energy collection, pollutant dispersal from buildings, the micro-climate around groups of buildings, flow around automobiles, design of high-lift devices and of low speed aircrafts, flow in engine and passenger compartments.

Also, three-dimensional incompressible flows are found in a wide range of practical engineering problems, such as in curved ducts, diffusers, and intake and exhaust of internal combustion engines, [5,6]. These problems can be attacked using analytical, experimental or numerical methods. Analytical solutions exist only for physically or geometrically simple problems. Experimental methods are expensive, the time requirement and effort associated with the model preparation and operation avoid extensive use. Computational fluid dynamics has recently become a third dimension in aerodynamics, complimenting the previously existing dimensions of pure experiment and pure theory. As the flow devices tend toward a more compact and highly efficient design, computational methods become an economical and time saving supplement to experiments for advanced analysis. Starting from mid 1960's, computational methods became a cost-effective design and analysis tool for the industry, [7].

The numerical solution of the 3-D Navier-Stokes equations to study large-scale problems has gained considerable attention in recent years. Discretization of the incompressible Navier-Stokes equations is often accomplished via the Finite Difference Method (FDM), The Finite Volume Method (FVM), The Residual Distribution Method or The Finite Element Method (FEM), [8-12]. FEM is considered to be more general and mathematically consistent approach, which can provide alternative possibilities for accuracy and cost adjustment, [13-15].

Demand for more accurate representation of the flow requires increased computational effort but it is unfortunately limited by the limited computational power of the day. This problem gave impetus to the study of computational algorithms to perform cost–effective analysis; the new idea is born, parallel computing. The size and complexity of the problems considered, necessitates the utilization of parallel computing, [16, 17].

Advances in both algorithms and computers have rapidly been absorbed by the CFD community in its quest for more accurate simulations and reductions in the time solution. Within this context, parallel computing has played an increasingly important role. The belief that parallel computing is the only way forward has remained undiminished. Moreover, the increasing reliability and acceptance of parallel computers has seen many commercial companies now offering parallel versions of their codes, [18].

Many problems in science and engineering are described by physical laws, formulated by equations. Computer simulation uses a discretized form of these equations. In CFD, the governing equations are nonlinear partial differential equations, describing the conservation principles for fluids in both space and time. In general, the domain of interest is very complex (irregular), e.g. it may be the outer region of an aircraft or a car. In order to resolve the flow field, grids comprising million points are needed when complex physical phenomena are encountered; for example, if turbulent flow problems are simulated. Moreover, to produce flow solutions that have design quality, certain accuracy has to be achieved, resulting in very large computing times. The realization of a computer windtunnel, apart from the difficulties in modeling the physical processes, can only be obtained by using the

most powerful massively parallel systems, [19,20]. Parallel processing is the present challenge to attain high performance computing; the recent technological growth in the hardware and software fields makes this challenge more realistic, by creating many expectations in the scientific community. CFD represents one of the research fields for which computing power requirements cannot be met by the present generation of supercomputers; since the accuracy level of the numerical flow simulation depends on the complexity of the mathematical model adopted and on the degree of resolution required, a large use of CFD for aerospace industry applications is strongly related to the development of new computing systems, parallel algorithms,[21,22].

Among all the ideas spawned by computer science over the last 20 years, none has transformed the field so profoundly as has parallel computation. In the late 1940s, a group of researchers at Princeton University proposed the first design of modern computer of today. This original design in which a computer consists essentially of a single processing unit, or processor, that executes a single sequence of data. The sequence of instructions is the program, which tells the processor how to solve a certain problem. In a parallel computer, by contrast, there are several processors, [16,17].

The advantage of parallel computers over classical vector supercomputers is scalability. They also use standard chips and are therefore cheaper to produce. Given a problem to be solved, it is broken into a number of subproblems. All of these subproblems are solved simultaneously, each on a different processor. In doing so, the processors may communicate with one another to exchange partial results. Finally, the results are combined to produce an answer to the original problem. This is a radical departure from the way the process of computation has been conducted over the past half century. The effective use of highly parallel machines introduces new aspects for algorithmic development.

Parallel computers are used primarily to speed up computations. A parallel algorithm can be significantly faster than the best possible sequential solution. Such fast solutions are truly needed. There is a growing number of applications-for example, in science, engineering, business and medicine-requiring computing speeds that can not

be delivered by any current or future conventional computer. These applications involve processing huge amounts of data, or performing a large number of iterations, or both, thus leading to inordinate running times. Parallel computation is the only approach known today that would make these computations feasible.

A number of criteria are commonly used in evaluating the goodness of an algorithm. The most important of these are the algorithm's running time, how many processors it uses, and the total number of steps it performs. A less widely used, but one of the important criteria is the algorithm's probability of success in completing the task.

A good parallel algorithm is the one, for which the ratio (speed-up) of the worst case running time of the fastest known sequential algorithm for the problem to the worst case running time of the parallel algorithm is large. For many computational problems, the largest possible speedup is at most equal to the number of processors used by the parallel computer, [16-17]. The efficiency of the algorithm must not deteriorate as the number of processors utilized increases. Scalability of an algorithm can be defined by considering its performance for either a fixed total problem size or a fixed problem size per processor. With the first definition, the parallel speed-up is limited by the inverse of the sequential portion of the algorithm. Thus, the parallel efficiency will tend to zero as the number of processors increases. The second definition is more relevant for distributed memory multiprocessors since the available total memory increases with the number of processors, [23].

The total efficiency can be expressed as a product of three factors termed *parallel*, *numerical* and *load balancing* efficiency. These factors describe: i. the increase in the number of floating point operations per grid node required to reach solution of the same accuracy when the number of subdomains is increased, ii. the increase of elapsed time for a parallel computation due to communication between processors during which computation can not take place, and iii. idle time of some processors due to uneven load.

For a chosen algorithm and communication pattern, the parallel efficiency can be predicted as a function of the computer parameters like latency time, communication speed, calculation speed, grid size and number of processors used. The parallel efficiency will increase substantially if the computation and communication can take

place simultaneously and one of the major factors affecting the effectiveness of parallelization is the numerical efficiency. The load balancing effects arise from the fact that in complex geometries the subdomains may not have the same size or the same boundary surface. Furthermore, complex boundary conditions may also cause delays. While both parallel and load balancing efficiencies can be analyzed theoretically as a function of the most important parameters, numerical efficiency can only be determined by numerical experiments, [24,25].

In theory, it is possible to predict the execution time of any parallel system given certain processor parameters, such as calculation speed and communication speed and communication rate, but, unfortunately, the relation between parallel speed and the number of processors is not a simple one. It is possible for a system to run slower on a large number of processors than it does on a small number of processors. This phenomenon is known as *speeddown*, [26]. Furthermore, on computers, which take advantage of memory caching, it is possible to obtain *superlinear* speedups (i.e. speedups greater than the number of processors). Thus, mathematical models of parallel speed are quite complicated, and require much validation, [27-30].

The use of networked workstations provides an easily accessible and cost effective alternative to supercomputers. In order to use this resource, methods based on domain decomposition, which is modified and applied in this thesis, were developed earlier, [31-38]. The computational domain is divided into several sub-domains and distributed among the workstations for concurrent processing. The sub-domains periodically exchange boundary information as the computation progresses. The global solution is then put together during post-processing. While domain decomposition-based parallel computing makes it possible to solve larger problems, the increased network traffic, due to exchange of boundary information of subdomains, presents a new problem. Parallelization of the code is based on dividing the solution domain into several subdomains or *blocks*, [31]. The global solution is then assembled by solving the equations for each block separately. The blocks are interconnected by means of *interfaces*. These are composed of nodes and elements that are shared by the two neighboring blocks. The numerical integration of equations are carried out in a *block solver*. A block solver and its interface solver are on the same processor. They are in fact, subroutines of the same program unit

running on a processor. The distribution of subdomains among a given number of processors can be optimized based on their computation and communication requirements, [31-37].

During the last years an increasing effort has been dedicated to the parallelization of scientific solvers, in the field of CFD, several parallel software packages solving either the Euler or the full Navier-Stokes equations around complex geometries have been developed and are currently used in production mode by aircraft or engine manufacturers. Most of the available parallel flow solvers have been developed by using the PVM (Parallel Virtual Machine) environment as *message passing* software; it is accepted as the standard software for distributed computing. It allows to join a set of heterogeneous hosts connected by a network, giving rise to a Virtual Machine: it appears to the user as a single large parallel computer. So a PVM application can be run across a set of different architectures at the same time. Besides, each PVM code is easily portable from an architecture to another, or a code written for a system can be compiled and executed on another one without modifications.

The later emergence is the MPI (Message Passing Interface) software. It is designed with the aim of creating a standard system that each vendor can optimize for his own architecture. It contains a set of point to point communication routines and a set of collective routines. MPI applications guarantee better parallel performances, because they use the native communication functions of each architecture, instead of the standard network communication functions. The portability to different architectures is preserved, but on the other side, the interoperability between MPI implementations on heterogeneous hosts is not allowed. PVM performs better than MPI when the communication time is much larger than the latency time and MPI performs better when the number of processors is high, due to the better latency, [38-40]. These software packages are available in public domain, which can be installed on cluster of workstations (WS) to communicate with each other. By this development, for the last ten years, the parallel computation has been applied in many areas successfully also in Turkey.

1.1. Purpose and Structure of the Report

The numerical solution of the 3-D Navier-Stokes equations to study large-scale problems has gained considerable attention in recent years. The purpose of this study is to write a parallel CFD code which solves the 3-D incompressible Navier-Stokes equations accurately and efficiently, using a new implicit method. It is also hoped to contribute towards the development of parallel algorithms. The scheme in discretizing the equations must be high-order accurate, fast and efficient, since memory requirements and computing times are important subjects to be considered. Explicit solution techniques are cheap per iteration and have simple algorithms, but total computation time is expensive. Implicit method requires expensive memory, but reduces the computation time efficiently. Because of the stability requirements, explicit schemes impose severe restrictions on the time step size for complex flows, which are generally resolved with sufficiently fine grids. Therefore, the implicit flow solvers are preferred in the study of such flows. In solving the viscous incompressible Navier-Stokes equations, the size of the problems necessitates the parallel computation. Therefore the parallel implementation of the solution method is an important task.

In implicit CFD methods, the governing differential equations are discretized in such a way that values at a new time level depend on neighboring values at the same time level. This technique is well known, and is described in detail, [27,41,42]. The end result is the need to solve a large set of simultaneous equations, which may be written in matrix form as:

$$A\varphi = B \tag{1.1}$$

where A is a sparse coefficient matrix, φ is the solution vector, and B is the source vector. A number of methods are available for solving equation (1.1), but it is not obvious which will be the fastest on a parallel computer. Solvers using explicit time-integration schemes have to use small time-steps for stability reasons. Typically a steady-state solution is reached after a large number of time-steps, resulting in a significant amount of time spent on interface communications. An implicit algorithm

can use larger time steps and because of the reduced number of time-steps to reach the solution, an implicit solver may outperform an explicit one in parallel computing by reducing the total amount of inter-block communication, [31-37].

Thus, in this study, the first step is taken by testing some algorithms via the sequential runs, considering the accuracy and computational time. This work and the results are explained in Chapter 6 and the most suitable one for the parallel processing is chosen before starting to write the parallel implicit CFD code. The implicit computational code has been modified in such a way that the solution is obtained with direct auxiliary potential function formulation for the pressure. Among the four methods tested via the sequential runs, the direct solution of auxiliary potential formulation has been observed to be more efficient in terms of computational time and easier programming, [1,43].

This thesis investigates a new implicit algorithm and also develops the new parallel algorithm for the implicit solution of the governing equations. Viscous incompressible flows have been studied with accurate parallel solution to Navier-Stokes equations in cluster of workstations operating in PVM environment. The space is discretized with brick elements via Galerkin Finite Element Method, while modified version of the two-step fractional method is used in time discretization of the momentum equation. Matching and non-overlapping grids are used. The scheme is second-order accurate in both time and space. This allows us to resolve the flow field with a smaller number of points while taking large time steps and so the size of the matrix to be inverted in the implicit method becomes small. At each time step, the momentum equation is solved only once to get the half time-step velocity field. The pressure at each time level is obtained via an auxiliary scalar potential which satisfies the Poisson's equation. Direct pressure calculation instead of using auxiliary potential function is also tested for the parallel performance.

For the parallel implicit solution of the matrix equations, modified versions of the Domain Decomposition Method is utilized for parallel solution of both the momentum and the auxiliary potential equations using non-overlapping matching sub domains. Direct inversion in each domain is performed. Both structured and

unstructured grids are used. The node numbering is specially arranged to obtain the minimum bandwidth so to save from computer storage and to have an advantage in terms of computational time.

The first results are obtained on a cluster of DEC Alpha XL266 workstations running Linux operating system, interconnected with a 100 Mbps TCP/IP network. Public version of the PVM 3.3 is used as the communication library. But the latest studies are developed on SGI Origin 3000 with 8 processor running Unix operating system.

In Chapter 2, the governing equations of the unsteady, incompressible viscous flow are provided. Chapter 3 describes the numerical formulation of the new implicit method and the direct solution for the half-time velocity and auxiliary potential function for the pressure field is defined in Chapter 4.

First implementation of the domain decomposition method and parallel programming is explained in Chapter 5. The four solution methods are tested in Chapter 6, to find the most efficient one for the parallel computations. Among the four methods, the direct auxiliary potential formulation is the best in terms of computational time. Sequential run results are compared with the cubic lid-driven cavity flow with Reynolds number 1000.

The new implicit, parallel implementation is calibrated with three-dimensional cubic cavity problem with Reynolds number of 1000, in Chapter 7. The first parallelization results on 2, 4 and 6-domain partitioning, having equal number of grid points, are discussed in detail. Having obtained the first results; it is seen that the speed-up is not good enough. Two more paralellization ways are tested by the runs on DEC Alpha XL266 workstations and comparisons of the three modifications are presented. Load balancing applied to increase the speed-up on the cluster of workstations is presented in this chapter. Ideal value of speed-up is obtained. The velocity and pressure profiles obtained accurately are presented also in the Chapter 7.

With this new development, the larger size cubic cavity problem is solved accurately. The size of the code decreased successfully to be able to run on larger size problems. As the stretching along the walls of the cavity is increased, more accurate results are obtained.

After having the benchmark solution data for three-dimensional lid-driven cavity flow with Reynolds number 400, parallel studies are continued with this Reynolds number at the end of Chapter 7, very good agreement with two benchmark solutions is shown. This chapter results that the parallel code works accurately but still needs development in terms of computational time and speed-up terms.

Chapter 8 lists the studies on SGI ORIGIN 3000 utilized with 8 processor running Unix operating system. The results are compared with the results obtained on DEC-Alpha cluster of workstations. Full geometry of the lid-driven cubic cavity is investigated at the symmetry plane and the results are compared with the old computations including only the symmetry part.

Chapter 9 continues the speed-up analysis. The direct pressure formulation is introduced and parallel solutions for the lid-driven cubic cavity problem is compared with the parallel results obtained via the auxiliary potential function formulation.

The domain decomposition equations are modified and with this new method, the computational time decreased efficiently and super-linear speed-up is obtained. At the end of Chapter 9, this method is introduced and compared with the previous parallel results obtained in this study.

Finally, Chapter 10 closes the work done for this thesis, by summarizing the results and by making conclusions.

Subroutines used for the direct solution of un-symmetric and symmetric systems are presented in Appendix A and B, respectively, the flow chart of the parallel implicit code (at one time-step) can be seen in Appendix C.

2. DEVELOPMENT OF THE NUMERICAL METHOD

In this study, the first step is taken by implementation of a second order accurate scheme (both in time and in space), which has been already developed, [44], for parallel solution of the three-dimensional incompressible Navier-Stokes equations involving high Reynolds number laminar flows about complex configurations. After the modifications in the implicit CFD code and sequential runs, the best technique suitable for the parallel solution is chosen and the three-dimensional parallel implicit solver is written for the incompressible unsteady Navier-Stokes equations. The finite element method (FEM) with an implicit scheme is used in the discretization of governing equations. For the time discretization of the momentum equation, the two-step fractional method, [45,46-50] is modified. At each time step, the momentum equation is solved implicitly.

The implicit computational code has been arranged in such a way that the solution is obtained with direct auxiliary potential function formulation for the pressure, instead of the element-by-element (EBE) iteration technique as employed in [51,52]. Iterative solution for the pressure equation at each time step constitutes the major difficulty in terms of parallel implementation. To overcome this burden, direct auxiliary potential function formulation explained in the proceeding chapters is implemented. The pressure at each time level is obtained via an auxiliary scalar potential which satisfies the Poisson's equation.

An iterative non-overlapping domain decomposition technique is implemented for parallel solution of the momentum and pressure equations. Different parallelization ways are tested, and lode balancing is applied, [53-55]. Poisson's equation is written both in terms of pressure and auxiliary potential and the parallel efficiency is analysed for both case. Matching and non-overlapping grids are used for solution of both the momentum and the auxiliary potential equations.

Explicit schemes are relatively easy to parallelize, since all operations are performed on data from preceding time steps. It is only necessary to exchange the data at the interface regions between neighboring subdomains after each step is completed. The sequence of operations and the results are identical on one and many processors. The most difficult part of the problem is usually the solution of the elliptic Poisson equation for the pressure, [1,45-55].

Implicit methods are more difficult to parallelize. While calculation of the coefficient matrix and the source vector uses only 'old' data and can be efficiently performed in parallel, solution of linear equation systems is not easy to parallelize. For example, Gauss elimination, in which each computation requires the result of the previous one, is very difficult to perform on parallel machines, [1].

The parallel 3D implicit solver is calibrated with the cubic lid-driven cavity flow with different Reynolds numbers. The node numbering is first made in x direction (along the direction of lid velocity in cavity problem) and then in y-direction (symmetry). Finally, the numbering is continued along the z-direction of the cavity. This kind of numbering of the nodes and elements enables us to save from computer storage, because by doing so, we obtain the minimum bandwidth, which also gives great advantage in terms of computational time.

One of the important properties of the parallel implicit code written in this study is that the ability of taking large time step size and of computing the results in a very short computational time, compared to the explicit results. The accuracy is conserved.

During the first half part of this work, the scheme here is made to run on cluster of workstations working in PVM environment. The cluster of workstations used is shown. There are 6 DEC-AlphaXL266, for number crunching and 2 DECAlpha166 WS for post and pre-processing. All communicates through a 10/100 Mbit Switching Hub. The studies are carried on SGI Origin 3000 Unix system with eight processors and the two systems are discussed.

Final studies are concentrated on the domain decomposition technique and this technique is modified in order to obtain the most efficient parallel solution.

2.1. Governing Equations

The Navier-Stokes equations are a system of second-order partial-differential equations and describe general fluid motion. The convective term in the momentum equation is non-linear, and therefore, numerical methods must in general be used to solve this system of equations. Because the momentum equations are vector equations, the contributions due to the viscosity (the diffusive terms) become a bit more complex and their treatment needs to be considered in more detail. The momentum equations also contain a contribution from the pressure. It may be regarded either as a source term (as body force) or as a surface force, but due to the close connection of the pressure and the continuity, it requires special attention.

Solution of the Navier-Stokes equations is complicated by the lack of an independent equation for the pressure, whose gradient contributes to each of the three momentum equations. Furthermore, the continuity equation does not have a dominant variable in incompressible flows. Mass conservation is a kinematic constraint on the velocity field rather than a dynamic equation, [1,44].

The major difference between the equations of compressible flow and those of incompressible flow is their mathematical character. The compressible flow equations are hyperbolic which means that they have real characteristics on which signals travel at finite propagation speeds. By contrast, the incompressible equations have a mixed parabolic-elliptic character. The difference in character can be traced to the lack of a time derivative term in the incompressible continuity equation

Adding the three forces, the viscous force $\mathbf{F}_{viscous}$, the pressure force $\mathbf{F}_{pressure}$ and the body force $\mathbf{F}_{body}$, and equating them to Newton's law for fluids yields the equation

$$\rho\frac{\partial \mathbf{u}}{\partial t}+\rho\,\mathbf{u}.\nabla\mathbf{u}=-\nabla p+\mu\nabla^2\mathbf{u}+\mathbf{F}_{body} \tag{2.1}$$

where ρ, $\mathbf{u}$=u(u, v, w), t, p and μ denote the density, the velocity vector, the time, the pressure and the dynamic viscosity, respectively.

As it must, the Navier-Stokes equations satisfy conservation of mass, momentum and energy. Mass conservation is included implicitly through the continuity equation. Conservation of momentum requires,

$$\frac{\text{momentum}}{\text{mass}} = [\text{mass}] + [\text{pressure}] + [\text{body force}] + [\text{viscosity}]$$

The Navier-Stokes equations with no body force (i.e., $\mathbf{F}_{body}=0$),

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}.\nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} \tag{2.2}$$

where $\nu = \mu/\rho$ is the kinematic viscosity. Equation (2.2) can be put into dimensionless form using the definitions

$$x' = \frac{x}{L},\ y' = \frac{y}{L},\ z' = \frac{z}{L},$$

$$\mathbf{u}' = \frac{\mathbf{u}}{U_o},\ p' = \frac{p - p_o}{\rho U_o^2},\ t' = \frac{tU_o}{L} \text{ and } \nabla' = L\nabla. \tag{2.4}$$

Here, U_o and L are a characteristic velocity and a characteristic length.

The equations used to model the flows considered within this project are the equations governing the flow of an unsteady, incompressible viscous fluid. Assuming constant, the non-dimensional form of the three dimensional, unsteady, incompressible Navier-Stokes equations can be written as

Continuity equation:

$$\nabla.\mathbf{u}' = 0 \tag{2.5}$$

and the momentum (Navier-Stokes) equation

$$\frac{\partial \mathbf{u}'}{\partial t} + \mathbf{u}'.\nabla'\mathbf{u}' = -\nabla' p' + \frac{1}{\mathrm{Re}}\nabla'^2\mathbf{u}'. \tag{2.6}$$

The equations are written in vector form (here on, boldface type symbols denote vector or matrix quantities).

The non-dimensional parameter Re is the Reynolds number based on the characteristic length L and velocity U_o. Pressure is a parameter fixed by the observer, so it follows that the only other force is inertia force. Furthermore, the relative magnitudes of the pressure and inertial forces are described by the Re, defined as

$$\mathrm{Re} = \frac{\mathbf{F}_{inertia}}{\mathbf{F}_{viscous}} = \frac{U_o L}{\nu} \tag{2.7}$$

For very small Reynolds numbers, the convective effects can be neglected and for high Reynolds numbers, the viscous effects are confined to a smaller layer close to the wall. Flow becomes turbulent at very high Reynolds numbers.

Hereafter, the superscript $'$ for non-dimensional quantities is omitted.

3. TIME AND SPACE DISCRETIZATIONS

3.1. Formulation

The governing equations are solved by a new version of the fractional step method proposed by Mizukami, [56], which is essentially based on Chorin's fractional step method, [45]. In Chorin's method, the essential boundary conditions for the normal component of the velocity must be embodied into the continuity equation, which leads to great difficulties.

In the present method, a potential function with a single degree of freedom at each node is introduced and Poisson equation for the potential is directly discretized, in which the essential boundary condition for the normal component of the velocity is treated as the natural boundary condition for the potential which still preserves the elliptic character of the physical problem.

For the time discretization of the governing equations, the following version of the fractional step method is chosen which is quite simple in algorithmic structure.

The weak form of the momentum equation (2.6) over the space-time domain reads as

$$\int_{\Omega t}\int \frac{\partial \mathbf{u}}{\partial t}.\mathbf{N}d\Omega dt = \int_{\Omega t}\int (-\mathbf{u}.\nabla\mathbf{u} - \nabla p + \frac{1}{\mathrm{Re}}\nabla^2\mathbf{u}).\mathbf{N}d\Omega dt, \qquad (3.1)$$

where N is an arbitrary weighting function.

Three steps are defined for the time discretization; half-time, intermediate-time and full-time steps.

The time integration of both sides of equation (3.1) for half a time step, $\Delta t / 2$, from old time level n to half-time level $n+\frac{1}{2}$ gives

$$\int_{\Omega}(\mathbf{u}^{n+1/2}-\mathbf{u}^{n}).\mathbf{N}d\Omega=\frac{\Delta t}{2}\int_{\Omega}(-\mathbf{u}^{n}.\nabla\mathbf{u}^{n+1/2}-\nabla p^{n}+\frac{1}{\mathrm{Re}}\nabla^{2}\mathbf{u}^{n+1/2}).\mathbf{N}d\Omega. \qquad (3.2)$$

At the intermediate-time step, if the intermediate-time step velocity is defined by $\mathbf{u}^{*}$, the time integration of equation (3.1), where the convective and viscous terms are taken at $n+\frac{1}{2}$ and pressure term at time level n, yields

$$\int_{\Omega}(\mathbf{u}^{*}-\mathbf{u}^{n}).\mathbf{N}d\Omega=\Delta t\int_{\Omega}(-\mathbf{u}^{n}.\nabla\mathbf{u}^{n+1/2}-\nabla p^{n}+\frac{1}{\mathrm{Re}}\nabla^{2}\mathbf{u}^{n+1/2}).\mathbf{N}d\Omega. \qquad (3.3)$$

For the full time step, the averaged value of pressure at old time level n and new time level n+1 is taken and the time integration of both sides of equation (3.1) for full-time step, Δt, is written as

$$\int_{\Omega}(\mathbf{u}^{n+1}-\mathbf{u}^{n}).\mathbf{N}d\Omega=\Delta t\int_{\Omega}(-\mathbf{u}.\nabla\mathbf{u}^{n+1/2}+\frac{1}{\mathrm{Re}}\nabla^{2}\mathbf{u}^{n+1/2}-\nabla\frac{p^{n}+p^{n+1}}{2}).\mathbf{N}d\Omega. \qquad (3.4)$$

Subtracting (3.3) from (3.4) results in

$$\int_{\Omega}(\mathbf{u}^{n+1}-\mathbf{u}^{*}).\mathbf{N}d\Omega=\frac{\Delta t}{2}\int_{\Omega}-\nabla(p^{n+1}-p^{n}).\mathbf{N}d\Omega. \qquad (3.5)$$

If one takes the divergence of equation (3.5), using the continuity equation (2.5), the following equation, which is discretized for the pressure, is obtained.

$$\int_{\Omega}\nabla.\mathbf{u}^{*}.\mathbf{N}d\Omega=\frac{\Delta t}{2}\int_{\Omega}\nabla^{2}(p^{n+1}-p^{n}).\mathbf{N}d\Omega. \qquad (3.6)$$

Subtracting (3.2) from (3.3) gives the intermediate-time step velocity,

$$\mathbf{u}^{*} = 2\mathbf{u}^{n+1/2} - \mathbf{u}^{n}. \tag{3.7}$$

3.2. Numerical Formulation

Various methods of discretization have from time to time been proposed both by engineers and mathematicians. All involve an *approximation* which, hopefully, is of such a kind that it approaches, as closely as desired, the true continuum solution as the number of discrete variables increases. The discretization of continuous problems has been approached differently, e.g. finite differences, weighted residuals.

The Finite Element method in science and engineering is applied to the differential equations with the aid of variational principles or the method of weighted residuals, [57]. In fluid dynamics there are a number of differential equations which do not have corresponding variational formulation or which have the variational formulation under certain restrictions only, [58,59].

The four types of weighted residual methods, the collocation, Galerkin, subdomain and least squares methods, are mostly preferred for finite element formulations since they are applicable to every differential equation. Among these methods, Galerkin and least squares methods are well adapted to finite element applications.

In this study, the Galerkin Finite Element method is used in order to evaluate the finite element matrix equations.

The Galerkin method, [43,60], is an orthogonal projection of any residual to a set of linearly independent complete functions (weighting functions). In order for any function to be the exact solution of the given equation, it is necessary for the residual to be zero, and this requirement, if the residual is considered to be continuous, is equivalent to the requirement of the orthogonality of the expression of residual to all the functions (weighting functions) of the system.

Let R denote the residual and W_i (i=1,m) the weighting functions, the Galerkin integral can be written as:

$$\int_{\Omega} R.W_i d\Omega = 0 \qquad (i=1,m) \qquad (3.8)$$

The weighting functions are expressed in terms of a finite number of constants, C_m, so that we can satisfy only *m* number of conditions of orthogonality. The determination of constants requires solving the constructed system of equations. Substituting the constants into the family of functions mentioned above gives the required approximate solution. The governing equations are transformed into finite element equations governing all isolated elements and these elements are finally collected together to form a global system of differential or algebraic equations with proper boundary and initial conditions. The nodal values of the variable are determined from this system of equations.

In this study, 8-node iso-parametric brick elements and trilinear interpolation (or so called shape functions) functions are used at each node. Let N_i (i=1,8) be the shape functions and *F* any differential equation, Galerkin finite element formulation is expressed as:

$$\int_{\Omega} N_i.F d\Omega = 0 \qquad (i=1,8) \qquad (3.9)$$

where, $d\Omega$ denotes the volume element of any finite element.

The weighting functions in Galerkin Method are the shape functions themselves. By multiplying the shape functions with the residual of the differential equation and integrating the product at the element level gives the orthogonal projection mentioned above. Setting the integral equal to zero, the element matrix equations are obtained.

To obtain the matrix equations for the governing equations, using the finite element discretization technique, following steps are taken.

According to the fractional step defined in this chapter, the half-time step velocity equation (3.2) can be written, using Galerkin Finite element technique, as:

$$\int_{\Omega} \mathbf{N}_i \mathbf{u}^{n+1/2} d\Omega = \int_{\Omega} \mathbf{N}_i \mathbf{u}^{n} d\Omega + \frac{\Delta t}{2} \left(\int_{\Omega} p^{n} \nabla \mathbf{N}_i d\Omega - \frac{1}{\mathrm{Re}} \int_{\Omega} (\nabla \mathbf{N}_i \nabla) \mathbf{u}^{n+1/2} d\Omega \right.$$

$$\left. - \int_{\Omega} \mathbf{N}_i (\mathbf{u}^{n} \nabla) \mathbf{u}^{n+1/2} d\Omega + \int_{\Gamma} \mathbf{N}_i \, h_{\alpha} d\Gamma \right) \tag{3.10}$$

where, h_{α} is the given boundary data and, $d\Gamma$ is the boundary of $d\Omega$.

The velocity is interpolated as follows:

$$\mathbf{u}^{n+1/2} = \sum_{i=1}^{e} \mathbf{N}_i \, \mathbf{u}_j^{n+1/2} \quad \text{and} \quad \mathbf{u}^{n} = \sum_{i=1}^{e} \mathbf{N}_i \, \mathbf{u}_j^{n} . \tag{3.11}$$

Substituting the interpolated values of velocities and the pressure value at the centroid of the element and integrating the equation over the element volume, the following matrix equations are evaluated:

$$\mathbf{M}_{ij} \mathbf{u}_j^{n+1/2} = \mathbf{M}_{ij} \mathbf{u}_j^{n} + \frac{\Delta t}{2} \left(\mathbf{B}_{\alpha i} + p_e^{n} \mathbf{C}_{\alpha i} - \frac{1}{\mathrm{Re}} \mathbf{A}_{ij} \mathbf{u}_j^{n+1/2} - \mathbf{D}_{ij}^{n} \mathbf{u}_j^{n+1/2} \right) \tag{3.12}$$

where the matrices are written as in the following:

$$\mathbf{M}_{ij} = \int_{\Omega} \mathbf{N}_i \, \mathbf{N}_j \, d\Omega , \tag{3.13a}$$

$$\mathbf{A} = \mathbf{A}_{ij} = \int_{\Omega} (\nabla \mathbf{N}_i . \nabla \mathbf{N}_j) d\Omega, \tag{3.13b}$$

$$\mathbf{B}_{\alpha} = \mathbf{B}_{\alpha i} = \int_{\Gamma} \mathbf{N}_i h_{\alpha} d\Gamma, \tag{3.13c}$$

$$\mathbf{C}_{\alpha} = \mathbf{C}_{\alpha i} = \int_{\Omega} \frac{\partial \mathbf{N}_i}{\partial \alpha} d\Omega, \tag{3.13d}$$

$$\mathbf{D} = \mathbf{D}_{ij}^{n} = \int_{\Omega} \mathbf{N}_i (\nabla \mathbf{N}_j) . (\mathbf{N}_k \mathbf{u}_j^{n}) d\Omega, \tag{3.13e}$$

where α indicates the Cartesian coordinate components x, y and z, the indices i, j and, k denote the element node number: the summation convention is employed on repeated indices (in this case 1 to 8), index *e* denotes the element number. **D** is the advection matrix, **A** is the stiffness matrix, $\mathbf{C}_{\alpha}$ is the coefficient matrix for pressure, $\mathbf{B}_{\alpha}$ is the matrix arising due to the boundary conditions.

The element matrix $\mathbf{M}_{ij}$ is the so-called mass matrix and simplified to the lumped mass approximation by row-sum at the element level so that the mass matrix can easily be inverted. The advection matrix **D** can be written as follows:

$$\mathbf{D} = \mathbf{D}_{ij}^{n} = \mathbf{u}_e^{n} . \mathbf{E}_{\alpha ij} \tag{3.14}$$

where $\mathbf{E}_{\alpha}$ is the matrix which arises due to incompressibility,

$$\mathbf{E}_{\alpha} = \mathbf{E}_{\alpha ij} = \int_{\Omega} \mathbf{N}_j \frac{\partial \mathbf{N}_j}{\partial x} \mathbf{u}_k^{n} \, d\Omega, \tag{3.15}$$

and the element velocity is found out using

$$\mathbf{u}_e^{n} = \frac{1}{vol(\Omega)} \int_{\Omega} \mathbf{N}_k \, \mathbf{u}_k^{n} \, d\Omega \ . \tag{3.16}$$

Finally, the matrix form of the momentum equation at the half-time step can be written as,

$$\left(\frac{2\mathbf{M}}{\Delta t}+\mathbf{D}+\frac{\mathbf{A}}{\mathrm{Re}}\right)\mathbf{u}_\alpha^{n+1/2}=\mathbf{B}_\alpha+p_e\mathbf{C}_\alpha+\frac{2\mathbf{M}}{\Delta t}\mathbf{u}_\alpha^{n}. \tag{3.17}$$

The system (3.17) is solved using a direct solution with the subroutine UNSYM (see Appendix A), which is oriented to systems with un-symmetric and banded coefficient matrix. So, the half-time step $\mathbf{u}^{n+1/2}$ velocity is obtained.

Using the above expressions, and defining the auxiliary potential function as

$$\phi=-\Delta t(p^{n+1}-p^{n}), \tag{3.18}$$

and choosing N as trilinear shape function, discretization of equation (3.6) reads as,

$$-\frac{1}{2}\mathbf{A}(p^{n+1}-p^{n})\Delta t=\frac{1}{2}\mathbf{A}\phi=2\mathbf{E}_\alpha\mathbf{u}_\alpha^{n+1/2}. \tag{3.19}$$

Finally, the system to be solved for the auxiliary potential function (3.18), which satisfies the Poisson equation, can be written as,

$$\mathbf{A}\phi=4\mathbf{E}_\alpha\,\mathbf{u}_\alpha^{n+1/2}. \tag{3.20}$$

where **A** is the symmetric and banded stiffness matrix.

The system (3.20) is solved using a direct solution with the subroutine SYM (see Appendix B), which is oriented to systems with symmetric and banded coefficient matrix. So, the auxiliary potential function, ϕ, is obtained at each time step implicitly.

The element stiffness matrix **A** is symmetric and constant at any time step. Therefore, once the global equations are solved, the assembled stiffness matrix **A** is factored and stored on disk.

Subtracting equation (3.3) from equation (3.4) and introducing the auxiliary potential function, ϕ, one obtains the following equation for the full-time step velocity field,

$$\mathbf{Mu}_{\alpha}^{n+1} = \mathbf{Mu}_{\alpha}^{*} + \frac{1}{2}\mathbf{E}_{\alpha}\,\phi\Delta t = 2\mathbf{Mu}_{\alpha}^{n+1/2} - \mathbf{Mu}_{\alpha}^{n} + \frac{1}{2}\mathbf{E}_{\alpha}\,\phi. \tag{3.21}$$

The element auxiliary potential ϕ_e is defined as

$$\phi_e = \frac{1}{\mathrm{vol}(\Omega_e)} \int_{\Omega_e} N_i\,\phi_i d\Omega_e, \quad i = 1,\ldots\ldots,8, \tag{3.22}$$

The element pressure p_e is obtained using equation,

$$p_e^{\,n+1} = p_e^{\,n} - \frac{\phi_e}{\Delta t}. \tag{3.23}$$

4. SOLUTION PROCEDURE

4.1. Direct Solution

The matrix systems for the half-time step velocity, auxiliary potential function and the full-time step velocity are introduced in Chapter 3 with the equations (3.17), (3.20) and (3.21), respectively. The solution of these systems of a large set of algebraic equations, resulting from the finite element formulations developed in Chapter 3, is important task, [43].

The equations (3.17) and (3.20) can be written as,

$$\mathbf{Ka} = \mathbf{r}\,, \tag{4.1}$$

where **K** is a coefficient matrix, a is a vector of unknown, and r is a vector of specified quantities. In this study, **K** associates with the un-symmetric coefficient matrix of half-time velocity in equation (3.17) or with the symmetric stiffness matrix in equation (3.20). **a** associates with the nodal half-time velocity in equation (3.17) or with the nodal auxiliary potential function in equation (3.20), and **r** associates with the load vector; the right-hand side of the equation (3.17) or the equation (3.20).

The coefficient matrix can be written as the product of a lower triangular matrix and an upper triangular matrix, i.e.,

$$\mathbf{K} = \mathbf{LU}\,. \tag{4.2}$$

This step is called the triangular decomposition of **K**. The algorithm for the triangular decomposition of an n x n matrix can be deduced as in Figure 4.1.

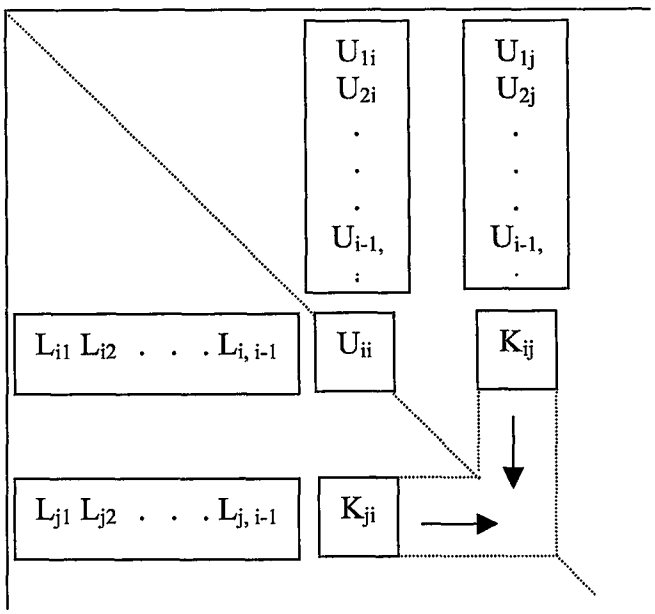


Figure 4.1 Triangular decomposition and reduction process

Triangular decomposition of the un-symmetric coefficient matrix of the half time velocity is computed once at each time step, the forward reduction of the load vector is done and the unknown half-time step velocity vector is solved by performing the back substitution. These steps are computed with UNSYM subroutine (Appendix A).

When **K** is symmetric, ($K_{ij} = K_{ji}$), as the stiffness matrix in equation (3.20), the interchanges of rows and/or columns are unnecessary in order to solve the equations. It is easy to verify in this case that

$$U_{ij} = L_{ji}U_{ii}. \tag{4.3}$$

For this problem, it is not necessary to store the entire coefficient array. It is sufficient to store only those coefficients above (or below) the principle diagonal. In this study, only the coefficients above the principle diagonal of the stiffness matrix are kept. This reduces by almost half the required storage for the coefficient array.

Half bandwidth is the important factor to reduce the coefficient array. More attention is paid for the element numbering of the computational domain, the most suitable element numbering, which will give the least size of half bandwidth, is found out at

the beginning of test problem computations. The storage method for symmetric banded stiffness matrix is shown in Figure 4.2. The upper triangular portion of the stiffness matrix is stored by columns. The columns are stored in single subscript array. This is done by YERSYM subroutine (APPENDIX B) once at the first time step. This method of storage is very easy to use vector dot product routines to effect the triangular decomposition and forward reduction, [43, 61-63]. It can be possible to reduce the required storage and computational effort still further by storing only the terms within a non-zero band.

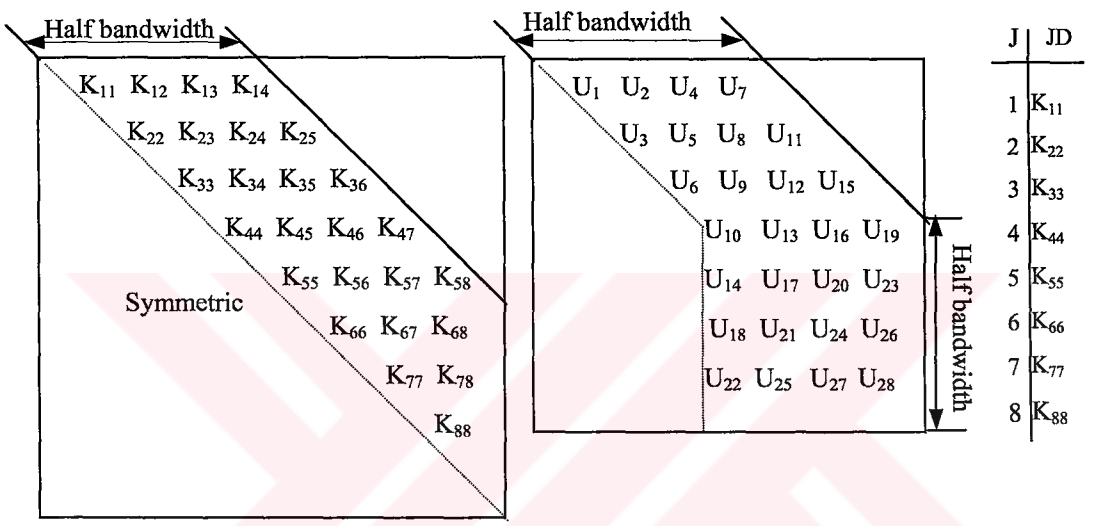


Figure 4.2 Banded storage of array

Triangular decomposition, forward reduction and the back substitution steps are computed at the SYM subroutine (Appendix B). The auxiliary potential function is solved directly by performing these steps at the SYM.

4.2 Computation Steps at One Time-step Level

With the initial boundary conditions for $\mathbf{u}^n$ and p^n, the following steps are taken to advance the solution one-time step:

i) The equation (3.17) is solved to find the half-time step velocity field, the parallel solution for the half-time step velocity field, $\mathbf{u}^{n+1/2}$, is obtained via the iterative domain decomposition method explained in Chapter 5.

ii) Knowing the half-step velocity field, equation (3.20) is solved for the auxiliary potential function, ϕ, which satisfies the Poisson's Equation. The parallel solution for the auxiliary potential function is obtained via the iterative domain decomposition method.

iii) With the auxiliary potential function ϕ, the full-time step velocity field, $\mathbf{u}^{n+1}$, is computed implicitly, using the equation (3.21).

iv) The associated pressure field p^{n+1} is determined implicitly, using the old time level pressure field p^{n} and the auxiliary potential ϕ obtained in step ii.

The above procedure is repeated until the desired time level. In all computations, lumped form of the mass matrix is used.

One of the most important characteristics of the scheme used in this thesis is that the stability of the computations does not depend on the time-step size, Δt. This is the basic advantage of implicit schemes to the explicit time stepping schemes. There is no restriction on the time-step size. Ability of taking the large time-step sizes, together with the parallel computation, perform very fast computational time for the complex and large size problems, this method has definite advantage over an explicit scheme.

4.3 Boundary and Initial Conditions

At a wall, the no-slip condition applies.

$$\mathbf{u}_{fluid} = \mathbf{u}_{solid} \tag{4.4}$$

For the auxiliary potential, ϕ is set to zero at the top wall. In the parallel computations, Neumann boundary conditions for velocity and auxiliary potential function are satisfied at the interfaces of the sub-domains.

5. FIRST PARALLEL IMPLEMENTATION

5.1. Introduction

The solution of large-scale CFD problems demands great computing resources. Parallel computation satisfies this demand and the use of parallel computers provides cost effective alternative to supercomputers. In order to use this resource, methods based on domain decomposition were developed earlier, [31-33, 51, 53-55].

In this thesis, a parallel program is developed for the solution of the unsteady incompressible 3D Navier-Stokes equations using the modified version of the fractional-step method with implicit time-integration scheme.

Domain decomposition technique, [64-67], is applied for the efficient parallel solution of the momentum equation (3.17) and the Poisson's Equation for the auxiliary potential function, equation (3.20). Because of the reduced number of time-steps, the implicit algorithm outperforms an explicit one in parallel computing by reducing the amount of communication. Direct solution of the momentum and auxiliary potential function (see Chapter 4) simplifies the parallel implementation and accelerates the computation.

5.2. Parallelization

During the half part of this study, parallel computations are processed on a cluster of DEC Alpha XL266 workstations running Linux operating system, interconnected with a 100 Mbps TCP/IP network. Public version of the PVM 3.3 is used as the communication library, Figure 5.1a. But the latest studies are developed on SGI Origin 3000 with 8 processor running Unix operating system, Figure 5.1b.

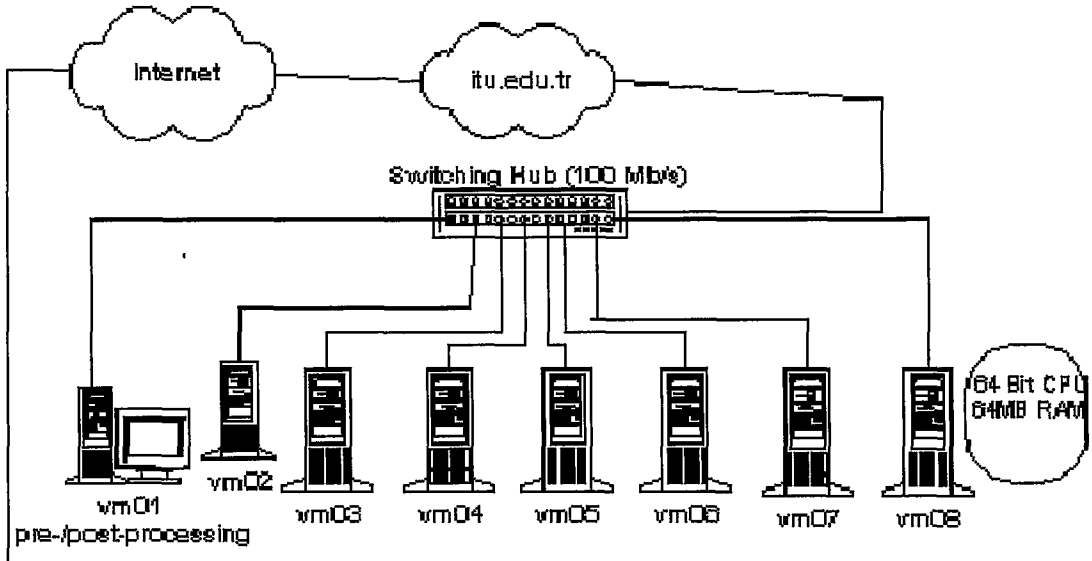


Figure 5.1a. Cluster of DEC Alpha XL266 workstations

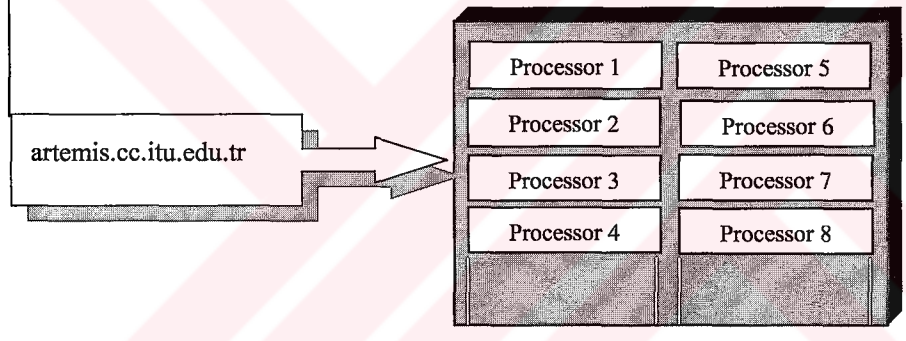


Figure 5.1b. SGI Origin 3000 with 8 processors

The computational domain is divided equally into 2, 4 or 6 sub-domains (*slaves*) and distributed among the workstations or processors for concurrent processing. Each domain Ω_i has the data of its grid and element numbering. Each domain knows its domain boundary $\partial\Omega_i$ nodes, its interface S_j nodes and its neighboring domains. Element and node numbering is done locally at each interface and the area of each element at the interfaces is calculated. Matching and non-overlapping grid is used.

Master processor spawns the slaves, gives their task number and their neighboring slaves task numbers at the beginning of the parallel program. Each slave knows the master. Pre-/post processing is computed by the master. The communications are set

up between the slaves and the master. The following steps of domain decomposition algorithm, [65], are taken as the parallel computation progresses.

5.2.1. Domain decomposition for the momentum equation:

In order to advance the solution single time-step, the momentum equation is solved implicitly and parallel using domain decomposition algorithm.

Initialization: Half-time step velocity equation (3.17) is solved with a direct solution (Chapter 4) separately in each domain Ω_i with boundary of $\partial\Omega_i$ and interface S_j, with vanishing Neumann boundary condition on the domain interfaces, Figure 5.2.

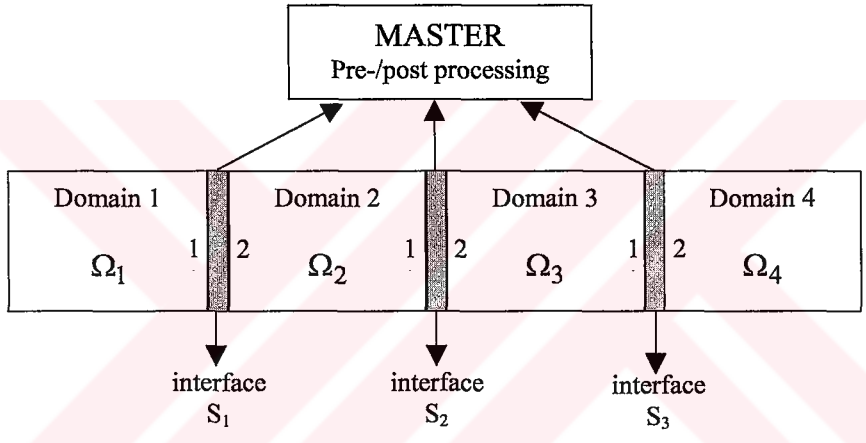


Figure 5.2. Parallel structure of 4-domain partitioning

Once the solution is obtained in each domain, each domain sends its interface nodal values to the master processor and master initializes the interface values. Master chooses an arbitrarily initial Neumann condition at the interfaces, (μ^o=0), and computes the difference between the received nodal unknowns at each interface, g^o, and sends back these differences, w^o, to the slaves. These procedures are shown in equations set (5.1). Here, $\overline{\mathbf{A}} = \frac{2\mathbf{M}}{\Delta t} + \mathbf{D} + \frac{\mathbf{A}}{Re}$ is the un-symmetric coefficient matrix of the half-time step velocity in equation (3.17), and $y_i = \mathbf{u}_\alpha^{n+1/2}$. The right-hand side of momentum equation (3.17) is represented by f_i.

$$\overline{\mathbf{A}}\mathrm{y}_\mathrm{i} = \mathrm{f}_\mathrm{i} \qquad \text{in } \Omega_\mathrm{i}, \qquad \mu^o = 0 \text{ (arbitrarily chosen)},$$

$$\mathrm{y}_\mathrm{i} = \mathrm{g}_\mathrm{i} \qquad \text{on } \partial\Omega_\mathrm{i}, \qquad \mathrm{g}^o = \mu^o - (\mathrm{y}_2^o - \mathrm{y}_1^o) S_j, \tag{5.1}$$

$$\frac{\partial \mathrm{y}_\mathrm{i}}{\partial \mathrm{n}_\mathrm{i}} = 0 \qquad \text{on } S_j, \qquad w^o = \mathrm{g}^o,$$

where subscripts i, and j denote the domain and interface numbers, respectively. Subscript o stands for the initial level at each time step. Subscripts 1 and 2 indicate the interface nodal value of the unknowns computed by the domains on the left and on the right sides of the interface, respectively.

With the initialized interface values, an iterative cycle begins at both slave and master processors. This cycle continues until the correct Neumann boundary conditions are obtained at the interfaces.

Unit Problem: Inside the interface iteration cycle, each slave starts to solve the unit problem using the direct solution, together with the initialized Neumann condition received from the master at the end of the initialization step, (equation set (5.2)).

$$\overline{\mathbf{A}}\mathrm{x}_\mathrm{i}^n = 0 \qquad \text{in } \Omega_\mathrm{i},$$

$$\mathrm{x}_\mathrm{i}^n = 0 \qquad \text{on } \partial\Omega_\mathrm{i}, \tag{5.2}$$

$$\frac{\partial \mathrm{x}_\mathrm{i}^n}{\partial \mathrm{n}_\mathrm{i}} = (-1)^{\mathrm{i}-1} w^n \qquad \text{on } S_j,$$

Each slave sends its new computed interface values to the master processor at the end of the unit problem and master starts post-processing of the received data from the slaves.

The interface iteration cycle, n, starts up at the master processor. Master optimizes the interface nodal values of the unknowns using the following '*steepest descent*' procedure.

Steepest Descent: Master processor computes the difference of the nodal unit problem values of the unknowns, x_1 and x_2, obtained at the interface by the two neighboring slaves. After the calculation of steepest descent coefficients, β and s, master tries to find out the optimum values at the interface which will give the correct interface solution for the original problem, (see equations set (5.3)). If the convergence condition, equation (5.4), is not satisfied, master sends w conditions (which is not converged) to the slaves and slaves start to solve the unit problem again with the new w condition at its own interface. Interface iteration cycle goes on until the correct Neumann conditions are obtained at the interfaces, μ^*.

$$aw^n = \left(x_1^n - x_2^n\right) S_j \qquad g^o = \left(y_1^o - y_2^o\right) S_j$$

$$g^{n+1} = g^n - \beta^n aw^n \qquad w^{n+1} = g^{n+1} + s^n w^n \tag{5.3}$$

$$\beta^n = \frac{\sum_j \int_{S_j} \left|g^n\right|^2 ds}{\sum_j \int_{S_j} (aw^n) w^n ds} \qquad s^n = \frac{\sum_j \int_{S_j} \left|g^{n+1}\right|^2 ds}{\sum_j \int_{S_j} (g^n)^2 ds}$$

$$\mu^{n+1} = \mu^n - \beta^n w^n$$

where n denotes the interface iteration level.

Convergence check: $\left|\mu^{n+1} - \mu^n\right| < \varepsilon \Longrightarrow \mu^{n+1} = \mu^k$ (5.4)

Finalization: Having obtained the correct Neumann boundary condition for each interface, the original problem is solved via the direct solution (Chapter 4) at each domain, together with the correct interface conditions, equations set (5.5).

$$\begin{aligned} \overline{\mathbf{A}}\mathbf{y}_i &= \mathbf{f}_i && \text{in } \Omega_i \\ \mathbf{y}_i &= \mathbf{g}_i && \text{on } \partial\Omega_i \\ \frac{\partial \mathbf{y}_i}{\partial \mathbf{n}_i} &= (-1)^{i-1}\mu^k && \text{on } S_j \end{aligned} \tag{5.5}$$

Solving equation (3.17) by parallel computation gives the velocity field at half-time step level, which is used at the right hand sides of Poisson's Equation (3.20) for obtaining the auxiliary potential.

5.2.2 Domain decomposition for Poisson's equation:

Using the velocity field at half-time level, the equation (3.20) is solved for the auxiliary potential function ϕ (which satisfies the Poisson's equation), in each domain Ω_i with boundary of $\partial\Omega_i$ and at interface S_j, with vanishing Neumann boundary condition on the domain interfaces. The domain decomposition steps indicated up to now for the momentum equation is repeated for the parallel solution of the system (3.20), but now the coefficient matrix $\overline{\mathbf{A}}$ is the symmetric stiffness matrix **A** in equation (3.20), $\mathbf{y}_i$ is the auxiliary potential function ϕ at each domain, and $\mathbf{f}_i$ is the right-hand side of the equation (3.20), the load vector. The solution of the auxiliary potential is obtained with domain decomposition where an iterative solution is also necessary at the interface. Therefore, the computations involving an inner iterative cycle and outer time step advancements have to be performed in a parallel manner on each processor communicating with the master processor.

Part of a flow chart concerning the master (parent) and slave (the child) processors are shown in Figure 5.3.

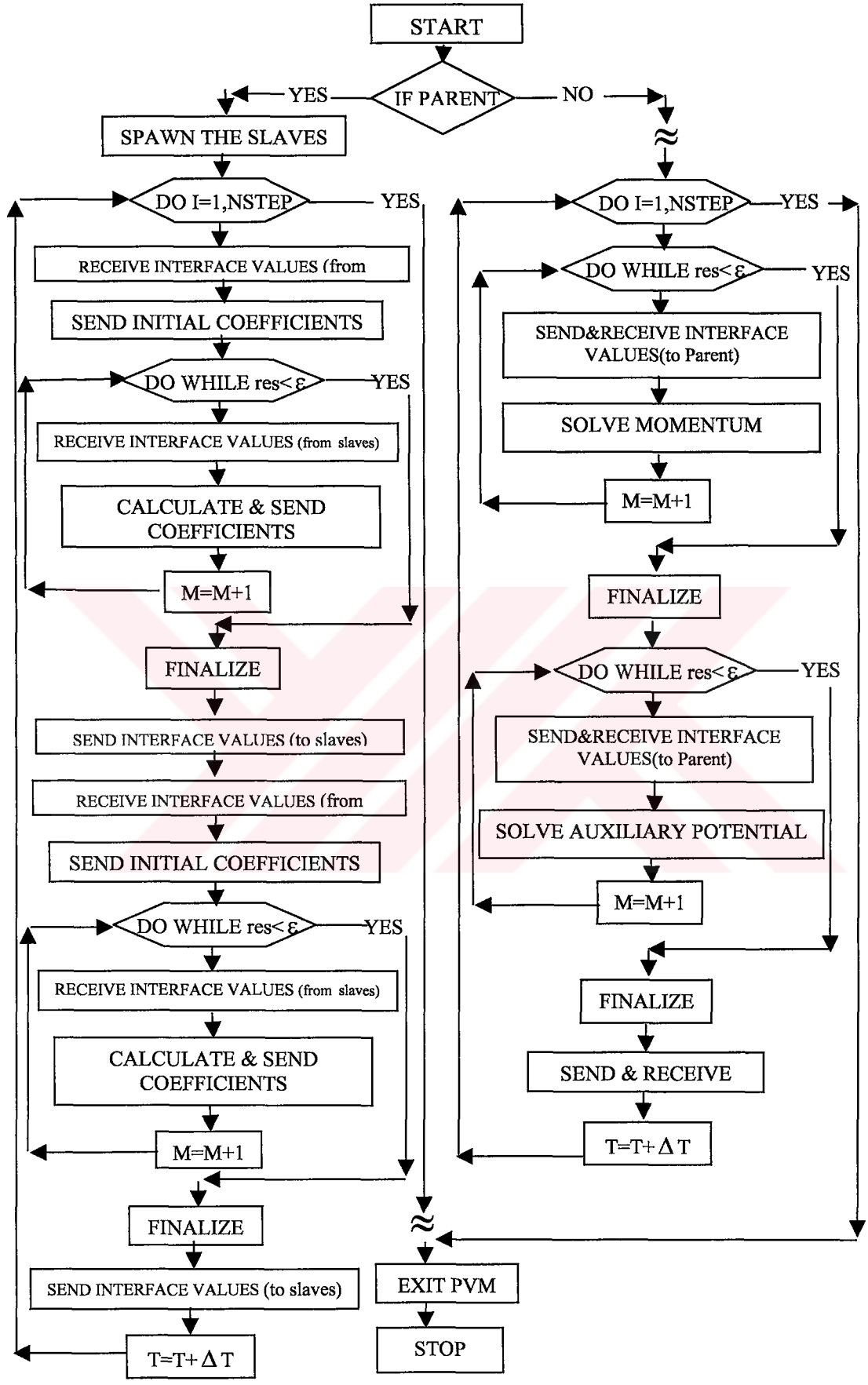


Figure 5.3. Flow chart of the first parallel implementation, concerning the parent and child processors.

6. TESTS ON THE IMPLICIT METHOD: SEQUENTIAL RUNS

6.1. Introduction

Before going on with the parallel programming, the first step is realized by some modifications on the implicit code and testing them via the sequential runs, considering the accuracy and computational time. The aim of this chapter is to check the solution method explained in Chapter 4 by sequential runs and to test four different algorithms and to find out the most suitable one for parallel programming, in terms of efficient computational time, accuracy and least computational effort.

To calibrate the results, a well-known laminar lid-driven cavity test problem is solved. The test case involves incompressible, laminar re-circulating flow and represent an important class of industrially interesting flows. The computations are carried on the DEC Alpha XL266 workstation running Linux operating system.

6.2. Three-Dimensional Lid-Driven Cavity Flow

The lid-driven cavity flow has received much attention over the past three decades. The incompressible flow in a cubic cavity whose top wall moves with a uniform velocity in its own plane has served as a model problem for testing and evaluating numerical techniques, in spite of the singularities at two of its corners. Interest in this problem is both practical and academic. The simple domain and boundary conditions leading to an enclosed shear-driven flow have made this an ideal benchmark system for validation of numerical algorithms. Although, the geometry is simple the flow is complex and rich enough to study fluid mechanics aspects. While the velocity and pressure fields in the lid-driven cavity have been computed using several numerical approaches, the flow stability in this system has yet to be thoroughly explored. The

flow is highly unsteady and possesses significant secondary motion in the spanwise direction and a complex three-dimensional pattern.

At the start of the upper wall motion, a primary recirculating flow exists due to the shear motion, generating a pressure-driven secondary flow in planes parallel to the end walls. The secondary flow establishes a spanwise circular pattern. Finally, Taylor-Görtler vortices develop from the interaction between the primary flow and the viscous end wall regions, [68].

Lid-driven flow is not only a benchmark problem in Fluid Dynamics, but also a problem having great application in real world as in the following examples. Many coating and lubrication processes have flow regimes similar to the lid-driven cavity hydrodynamics. Another example is that, a cavity is brought about in the abdomen of the F117 in the process of missile launching or bombing and vortices spawn in the cavity. Hence, the lift and drag will be affected by the vortices induction. The flow that subsequently develops in the cavity exhibits many features that are of interest to the aeronautical engineer, e.g. fluctuating pressure levels, localized heating/cooling, and recirculating flow regions.

A number of numerical reference solutions exist for the laminar lid-driven cavity test case, though it is not easily reproducible experimentally, [69]. On the other hand, most research has been focused on 2-D problems, [70]. Limited numbers of reliable numerical results for 3-D cavity flows have been obtained in the past several years.

6.2.1. Computational Details

Lid-driven flow in a cubic cavity with a Reynolds number of 1000 is selected as a test case. Cubic cavity has sides of length 1.0. The upper wall of the domain ($z = 1$) is set to move at a constant speed of 1.0 in positive x-direction. The Reynolds number is based on the lid length and the lid velocity. The geometry of a unit cubic cavity is shown in Figure 6.1. In the y-direction, flow is assumed symmetric with respect to the mid-plane ($y = 0.5$).

No-slip boundary conditions are employed on the solid walls of the cavity. The initials conditions are zero velocity everywhere except the lid. The Drichilet

boundary condition for the horizontal (positive x-direction) velocity, u, and auxiliary potential function, ϕ, is imposed at the lid nodes (z = 1) as u = 1, and ϕ = 0, respectively.

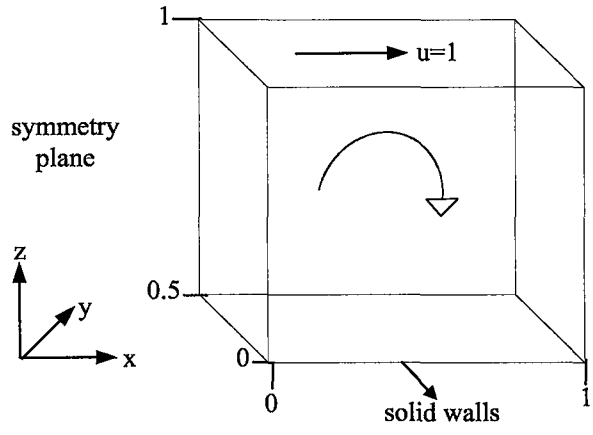


Figure 6.1. Definition of 3-D lid-driven cavity problem

For the tests on the implicit CFD program, the results are obtained with different computational domains prepared with 21x11x21 and 25x13x25 nodes, as shown in Figure 6.2 and Figure 6.3. Mesh is stretched near the walls of the cubic cavity.

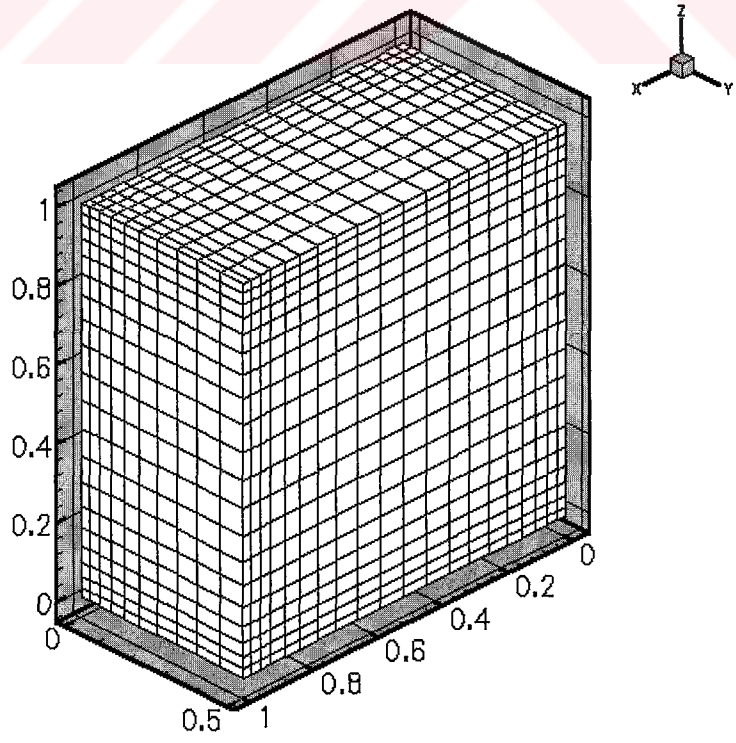


Figure 6.2. Domain with 21x11x21 nodes

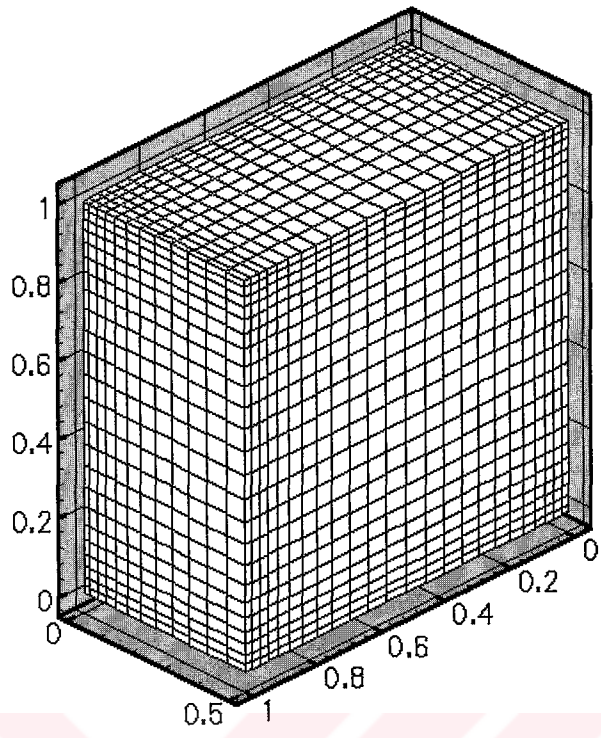


Figure 6.3. Domain with 25x13x25 nodes

The solution is advanced up to the dimensionless time level of 25, where the steady state is reached, with time-step size of 0.1. Minimum grid size is 0.024 for 21x11x21 grid and 0.020 for 25x13x25. Half-bandwidth of the symmetric coefficient matrix is equal to 244 and 340 for 21x11x21 and 25x13x25 nodes, respectively.

6.3. The Methods Tested by the Sequential Runs

The following 4 methods are compared by sequential computations in order to find the most suitable one for the parallel implementations:

METHOD 1: Direct auxiliary potential function formulation:

This is the original method implemented in this study, as described in Chapter 3 and Chapter 4. Before starting the parallel implementation, direct auxiliary potential function formulation is tested in terms of accuracy, computational time and effort.

The results obtained for the velocity vector field and for the pressure field are given in Figure 6.4 for 25x13x25 nodes.

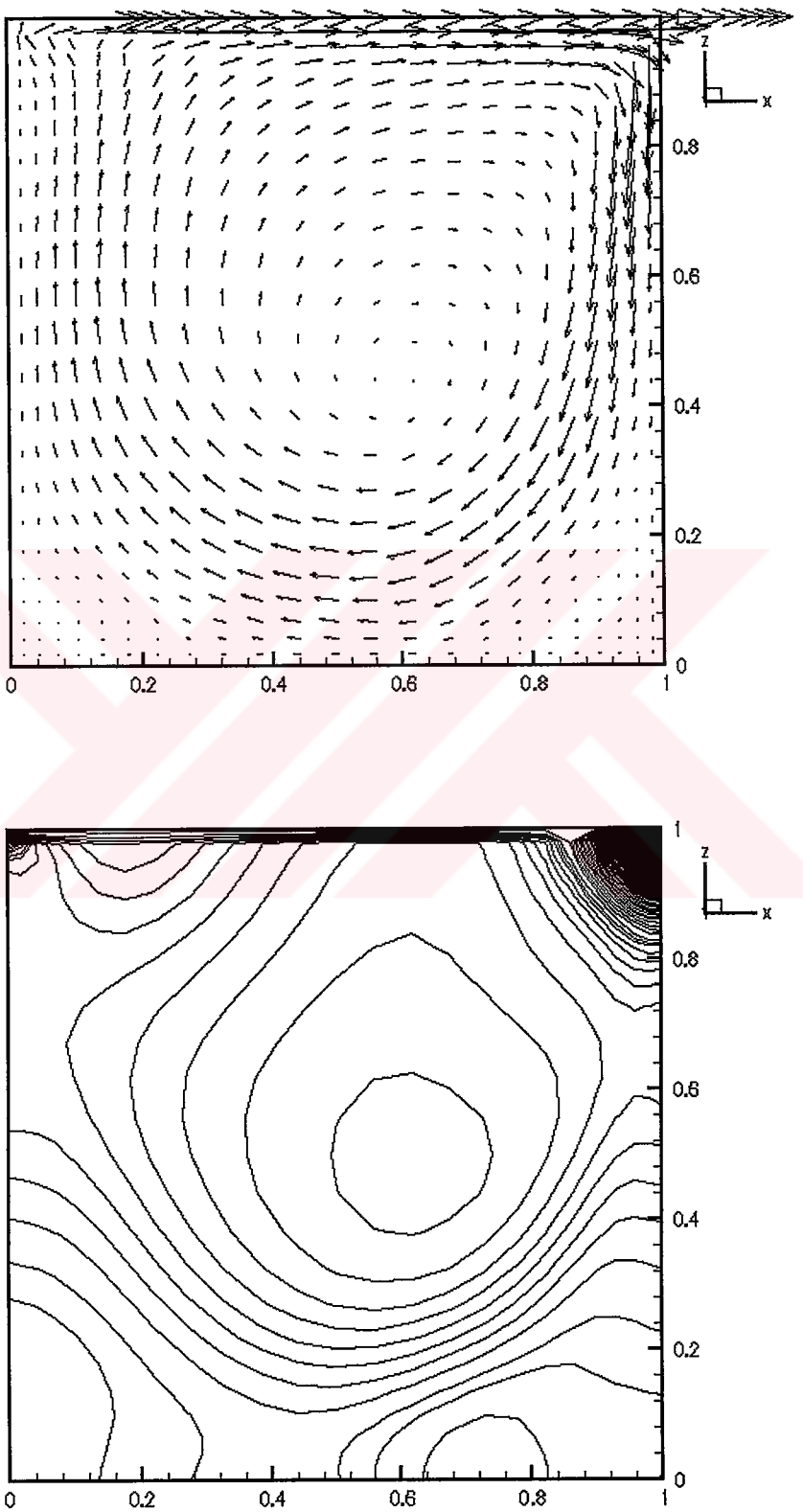


Figure 6.4. The velocity & pressure field obtained with method 1, sequential run, Re=1000, 25x13x25 nodes

Velocity field and pressure contours can be seen in Figure 6.5a and 6.6a, respectively, for 21x11x21 nodes.

The first result obtained for this method, which is chosen the most efficient one for the parallel computations, is compared with the benchmark solution computed with the pseudo-spectral method of Ku *et. al.*, [71]. Horizontal velocity at vertical centerline on symmetry plane is plotted for 21x11x21 domain, but this plot, Figure 6.7, shows that the fluid dynamics code was not working properly. Investigations detected the following errors in the modified implicit CFD program using method 1:

i) After the first time step, the symmetric stiffness matrix **A** in equation (3.20) was being zero instead of being constant.

ii) The boundary condition was imposed into the matrix **A** in a wrong way, so the structure of **A** was being destroyed.

iii) Full-time step $\mathbf{u}^{n+1}$ velocity calculation was wrong, $\frac{1}{2}$ coefficient of the last term on the right hand side of equation (3.21) was forgotten.

Following the corrections, the plot of horizontal velocity at vertical centerline of the symmetry-plane, Figure 6.7, shows the method 1, working properly. Plots shown in Figure 6.4 for 25x13x25 grid are drawn after the corrections.

METHOD 2: Iterative pressure formulation:

This is the method, which had been implemented already in reference [4,5]. Auxiliary potential function is calculated iteratively via the element-by-element iteration technique at each time step. As mentioned in the references, 'iterative solution for the pressure equation at each time step constitutes the major difficulty in terms of parallel implementation'. Iterations continue until the convergence criterion, which is set to 10^{-4}, is met.

Figure 6.5b and Figure 6.6b show the velocity field and pressure isolines, respectively, obtained by this method with 21x11x21nodes.

METHOD 3:The advection velocity is obtained at n+1/2 time step instead of n:

The only difference of this algorithm from the method 1 is explained by the following equations. Change of advection velocity affects all the momentum and auxiliary potential function equations. By taking the advection velocity at half-step time level n+1/2 instead of old time level n, the equation (3.2), in Chapter 3, becomes,

$$\int_{\Omega}(\mathbf{u}^{n+1/2}-\mathbf{u}^{n}).\mathbf{N}d\Omega=\frac{\Delta t}{2}\int_{\Omega}(-\mathbf{u}^{n+1/2}.\nabla\mathbf{u}^{n+1/2}-\nabla p^{n}+\frac{1}{\mathrm{Re}}\nabla^{2}\mathbf{u}^{n+1/2}).\mathbf{N}d\Omega. \quad (6.1)$$

At the intermediate time step, equation (3.3) is re-written as,

$$\int_{\Omega}(\mathbf{u}^{*}-\mathbf{u}^{n}).\mathbf{N}d\Omega=\Delta t\int_{\Omega}(-\mathbf{u}^{n+1/2}.\nabla\mathbf{u}^{n+1/2}-\nabla p^{n}+\frac{1}{\mathrm{Re}}\nabla^{2}\mathbf{u}^{n+1/2}).\mathbf{N}d\Omega. \quad (6.2)$$

For the full time step,

$$\int_{\Omega}(\mathbf{u}^{n+1}-\mathbf{u}^{n}).\mathbf{N}d\Omega=\Delta t\int_{\Omega}[-(\mathbf{u}.\nabla\mathbf{u})^{n+1/2}+\frac{1}{\mathrm{Re}}\nabla^{2}\mathbf{u}^{n+1/2}-\nabla\frac{p^{n}+p^{n+1}}{2}].\mathbf{N}d\Omega. \quad (6.3)$$

Subtracting equation (6.3) from equation (6.1) gives,

$$\int_{\Omega}(\mathbf{u}^{n+1}-\mathbf{u}^{*}).\mathbf{N}d\Omega=\frac{\Delta t}{2}\int_{\Omega}-\nabla(p^{n+1}-p^{n}).\mathbf{N}d\Omega., \quad (6.4)$$

where intermediate-time step velocity $\mathbf{u}^{*}$ is inter-calculated explicitly:

$$\mathbf{u}^{*}=\mathbf{u}^{n}+\Delta t\int_{\Omega}(-\mathbf{u}^{n+1/2}\nabla\mathbf{u}^{n+1/2}-\nabla p^{n}+\frac{1}{\mathrm{Re}}\nabla^{2}\mathbf{u}^{n+1/2}).\mathbf{N}d\Omega. \quad (6.5)$$

Taking the divergence of equation (6.4) gives $\mathbf{E}_{\alpha}\mathbf{u}^{*}=-\frac{\Delta t}{2}\mathbf{A}(p^{n+1}-p^{n})$.

If the auxiliary potential function is defined in terms of pressure as $\phi = -\Delta t(p^{n+1} - p^n)$, and is set in to equation (6.6), the following auxiliary potential function formulation can be obtained.

$$\mathbf{A}\phi = 2\mathbf{E}_\alpha \mathbf{u}^* \tag{6.7}$$

Finally, the full-time step velocity field can be obtained using,

$$\mathbf{M}\mathbf{u}_\alpha^{n+1} = \mathbf{M}\mathbf{u}^* + \frac{1}{2}\mathbf{E}_\alpha \phi \tag{6.8}$$

This method has also been tested with the 3-D cavity problem using 21x11x21 grid, and the velocity vector field and the pressure iso-lines are presented in Figure 6.5c and Figure 6.6c, respectively.

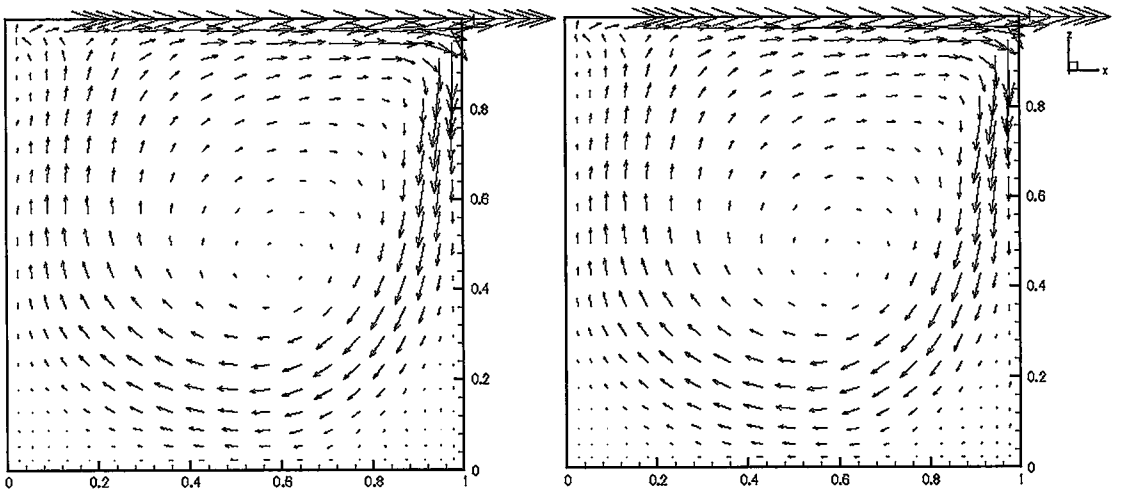
METHOD 4: $\left|\mathbf{u}^{n+1/2} - \mathbf{u}_*^{n+1/2}\right| < \varepsilon$ **where** $\varepsilon = 10^{-4}$**:**

In this method, the finite element formulation is exactly same with the formulation described in Chapter 3, the difference is, half-time step velocity, $\mathbf{u}^{n+1/2}$, is tried to be found out more accurately by iteration at each time step. Iterations go on until the arbitrarily chosen convergence criteria, ε, is met.

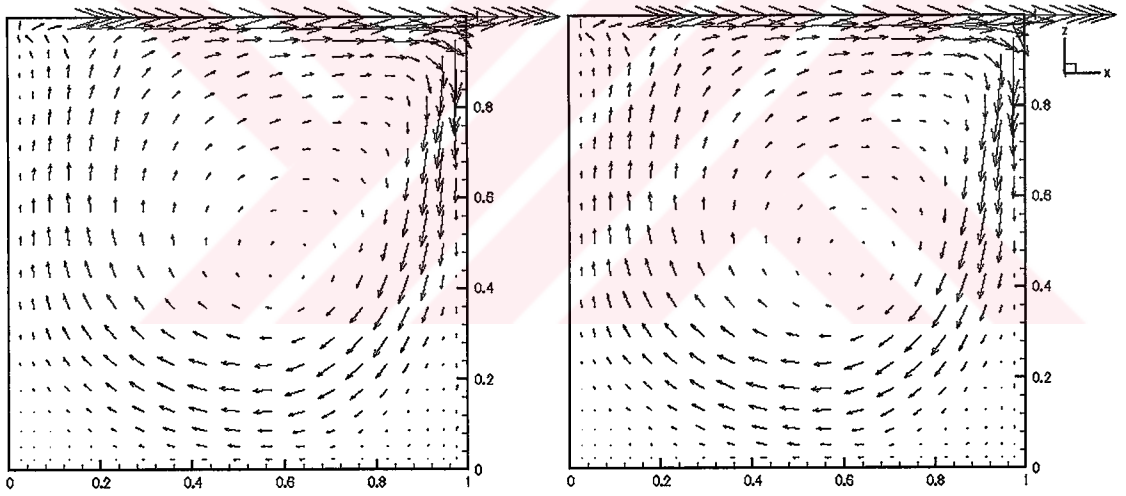
$$\left|\mathbf{u}^{n+1/2} - \mathbf{u}_*^{n+1/2}\right| < \varepsilon = 10^{-4},$$

where $\mathbf{u}_*^{n+1/2}$ is the old iteration value of half-time step velocity, $\mathbf{u}^{n+1/2}$.

The results of this method for the cavity problem on the domain 21x11x21 are shown in Figure 6.5d and Figure 6.6d, respectively.



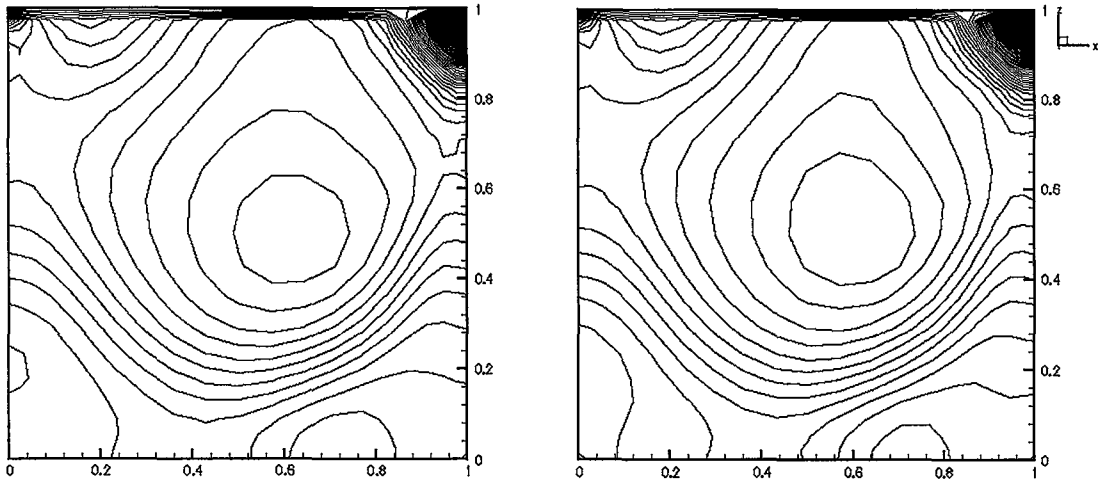
(a) METHOD 1 (b) METHOD 2



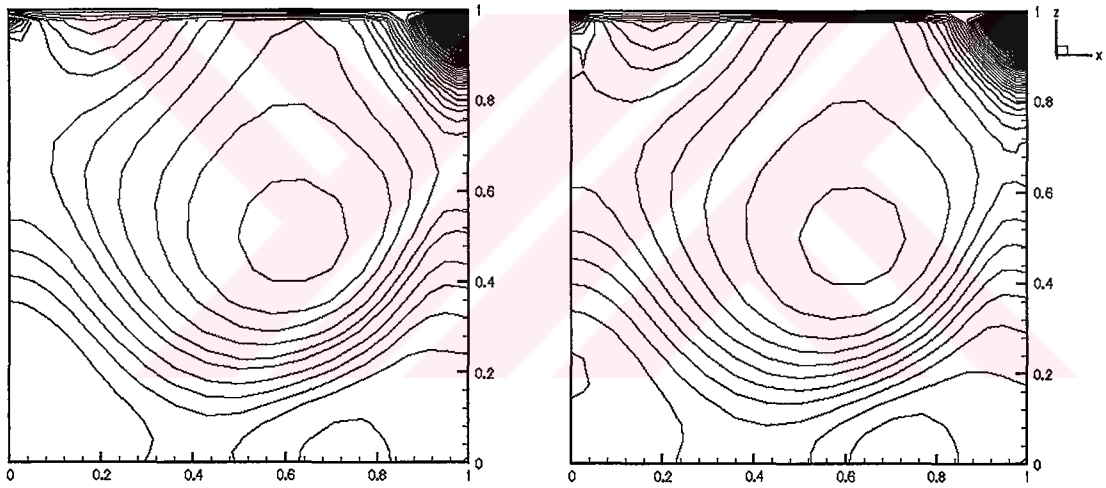
(c) METHOD 3 (d) METHOD 4

Figure 6.5. Velocity field plots for the tested methods on the symmetry-plane, Re=1000, 21x11x21 nodes, sequential run.

The results are obtained on DEC Alpha XL266 workstation, dimensionless time is 25 where $\Delta t = 0.1$.



(a) METHOD 1 (b) METHOD 2



(c) METHOD 3 (d) METHOD 4

Figure 6.6. Pressure contours for the tested methods on the symmetry-plane, Re=1000, 21x11x21 nodes

The results are obtained on DEC Alpha XL266 workstation, dimensionless time is 25 where $\Delta t = 0.1$.

6.4. Results and Discussion

As shown in Figure 6.5 and Figure 6.6, the moving lid drives a re-circulating flow in the 3-D cavity, convection is dominant and the primary vortex is offset towards the geometric center of the cavity.

Horizontal velocity at vertical centerline and vertical velocity at horizontal centerline on symmetry-plane, at steady state, is plotted in Figure 6.7 and Figure 6.8, respectively, and the methods are compared with the benchmark solution in reference Ku *et.al.*, [71]. The results show good agreement with the benchmark solution after the corrections are made for method 1, as explained in this chapter.

Velocity field comparison results show that, all of the methods give good agreement with the benchmark solution, although the grid 21x11x21 is coarse, and stretching is much more loose than the one in reference Ku *et.al.*, [71]. The results are obtained with a large time-step size as $\Delta t = 0.1$. In terms of computational time, the computations for the four methods, (on DEC Alpha XL 266 workstation), give different results as presented in Table 6.1.

Table 6.1. Comparison of CPU times of the four methods tested with sequential run

METHOD	CPU time (s)	ELAPSED time (s)
1	3733	3738
2	4431	4435
3	4135	4138
4	8521	8513

Table 6.1 shows that, method 1 (explained in detail in Chapter 3 and in Chapter 4) converges in a shortest computational time and has least computational effort among the other tested methods. These are the basic advantages of method 1 and make it the most suitable one among the others, for the parallel implementation (Chapter 5). The test case parallel results are going to be presented in the following chapters.

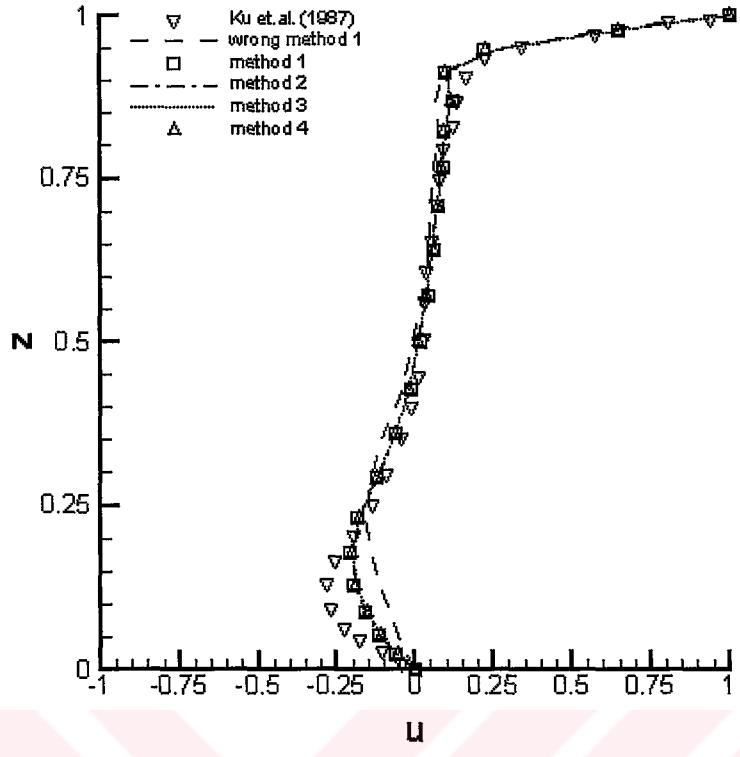


Figure 6.7. Horizontal velocity at vertical center-line, comparison of the methods, Re=1000, $\Delta t = 0.1$, 21x11x21nodes

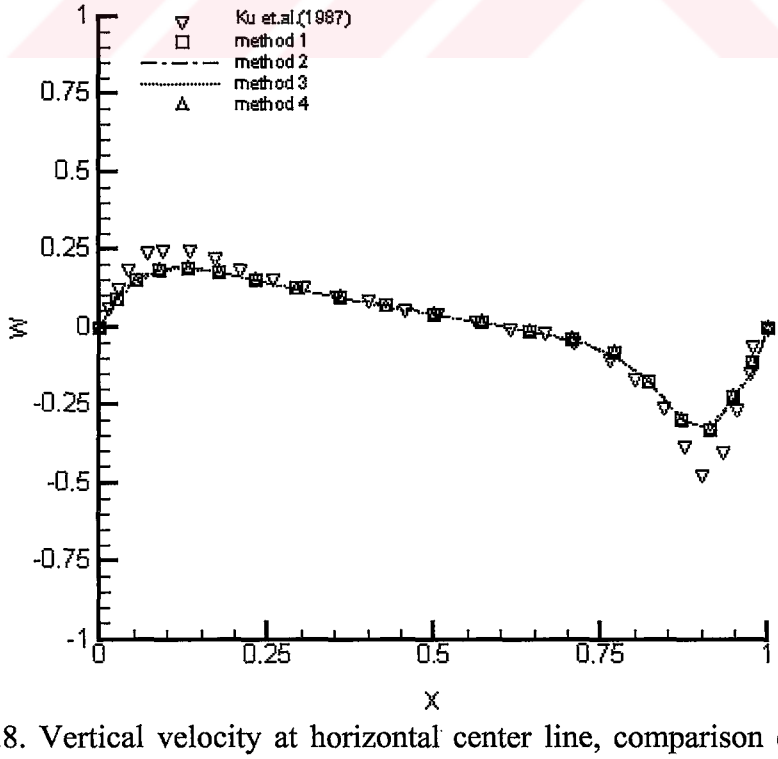


Figure 6.8. Vertical velocity at horizontal center line, comparison of the methods, Re=1000, $\Delta t = 0.1$, 21x11x21nodes

The horizontal u and vertical w velocities, along the centerlines of the symmetry-plane, y = 0.5, are also plotted for 21x11x21 and 25x13x25 grids in Figure 6.9 and compared with the benchmark solution of Ku *et.al.*, [71]. The results show that, with a finer grid and with a more stretched mesh near the walls, benchmark agreement can be increased.

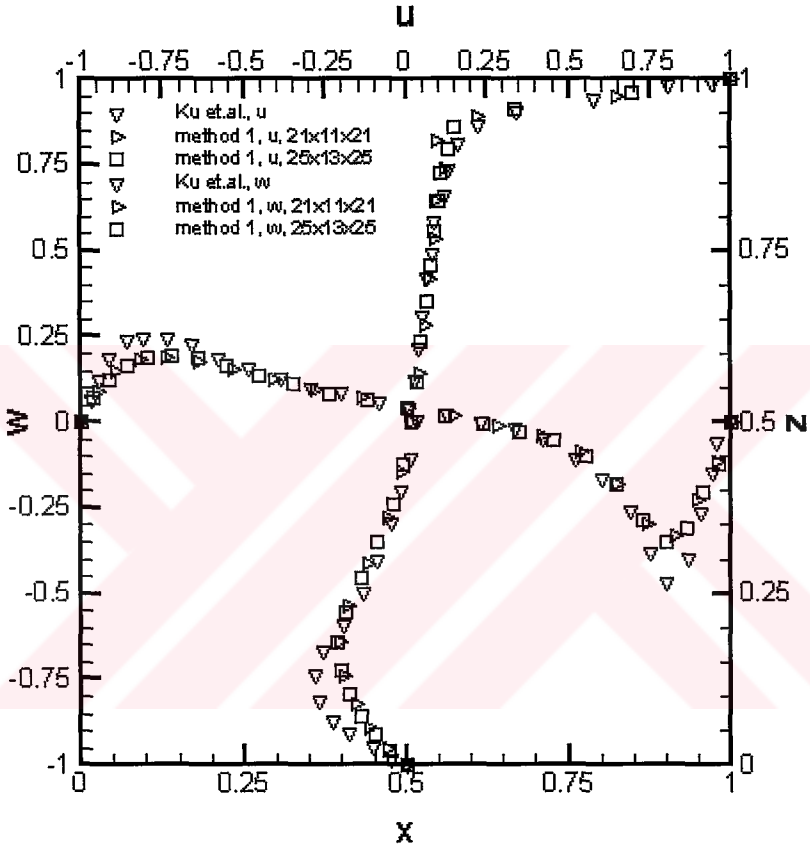


Figure 6.9. Velocity distributions along horizontal and vertical center-lines on the symmetry-plane of lid-driven cavity flow, method 1 result, sequential run, 21x11x21 & 25x13x25 nodes, Re=1000, $\Delta t = 0.1$.

7. PARALLEL IMPLEMENTATION RESULTS & DISCUSSION

7.1. Three-dimensional Lid-driven Cavity at Re= 1000

Following the researches presented in Chapter 6, direct solution method described in Chapter 3 and Chapter 4 is implemented for parallel computations and the first domain decomposition implementation is done as described in Chapter 5 (see Figure 5.3). Three dimensional lid-driven cavity test problem defined in Chapter 6 is chosen for calibration of the parallel implicit CFD program written in this study. Parallel results are obtained successfully, first on the 2-domain partitioning with 2 x (11x11x21) nodes, but for further parallel computations, cavity grid is prepared with total number of 25 x 13 x 25, to be able to get 6-domain partitioning results. The domain is divided into 2, 4 and 6 sub-domains. Node, element, interface and boundary nodes data are prepared for each domain and distributed on DEC Alpha XL266 workstations (see Chapter 5). 2x(13x13x25) for 2 domains, 4 x (7x13x25) for 4 domains and finally 6 x (5x13x25) nodes for 6 domains exist as shown in Figure 7.2, Figure 7.3 and Figure 7.4, respectively.

Global node and element numbering is first made along the positive x direction (along the lid), then in y-direction (symmetry) and finally along the z-direction of the cavity, Figure 7.1.

This way of numbering of the nodes and elements enables us to save from computer storage, because by doing so, we obtain the minimum bandwidth, which also gives great advantage in terms of computational time. The half-bandwidths for 2-, 4- and 6- partitioning are 184, 100 and 72, respectively. Decrease of half-bandwidth from 2-domain partitioning to 4-domain is much more than the decrease to 6-domain partitioning. If eight processors could be available, the more efficient computational time and speed-up results could be expected with 8-domain partitioning.

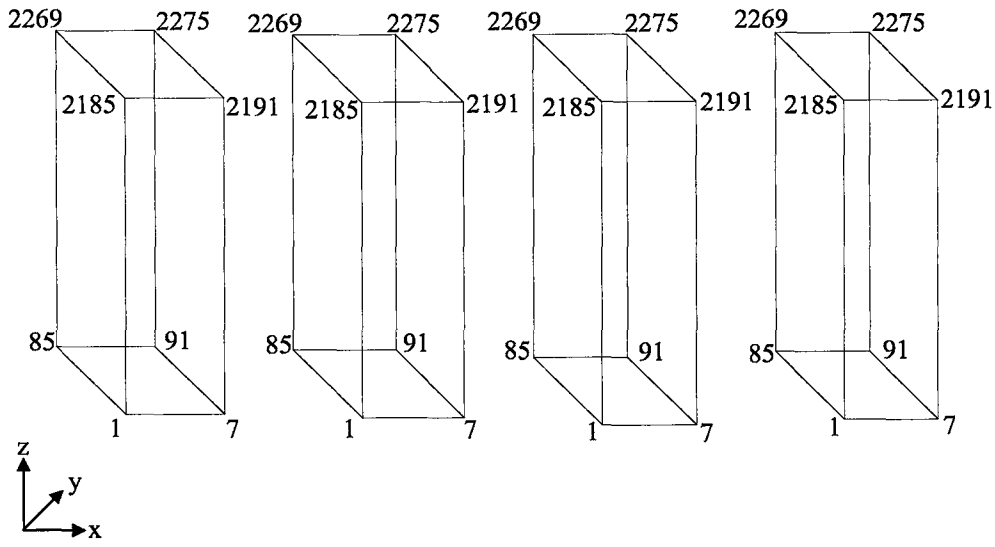


Figure 7.1. Global node numbering of each domain, 4-domain partitioning

Number of test runs is done to find the best convergence check criteria $\varepsilon_{\mathbf{u}}$ and ε_{ϕ} for the velocity and auxiliary potential interface iterations, respectively. The optimum results for 2-domain and 4-domain partitioning could be obtained successfully with $\varepsilon_{\mathbf{u}}=10^{-4}$ and $\varepsilon_{\phi}=10^{-6}$ but not for 6-domain computation. To be able to compare computational time of each parallel solution and to make speed-up analysis, the same convergence criteria should be taken for 2, 4 and 6 sub-domain computations. The best criteria for the domain decomposition method described in Chapter 5 are found out as

$$\varepsilon_{\mathbf{u}}=10^{-5} \text{ and } \varepsilon_{\phi}=10^{-6},$$

and applied at all parallel runs. Although the interface iterations and so the computational time decreases with looser convergence criteria, stability of the results could be damaged, accuracy could get worse. Special care should be taken for the interface u, v, w velocities and auxiliary potential function calculations, at each parallel run.

The solution is advanced up to the dimensionless time level of 25, where the steady state is reached, with time step of 0.1 and for minimum grid size of 0.019.

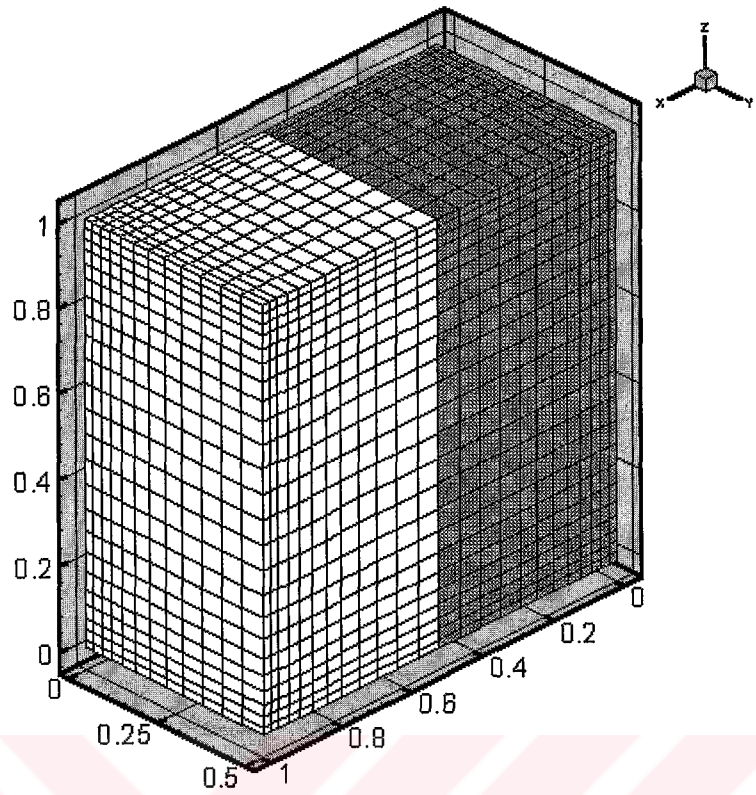


Figure 7.2. Cavity grid, 2-domain partitioning, 2x(13x13x25) nodes

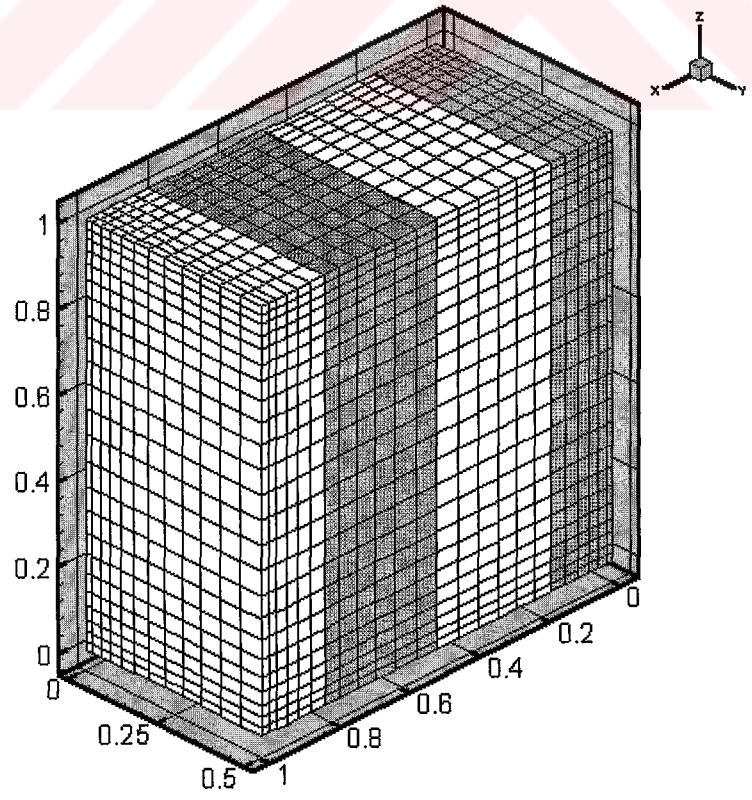


Figure 7.3. Cavity grid, 4-domain partitioning, 4x(7x13x25) nodes

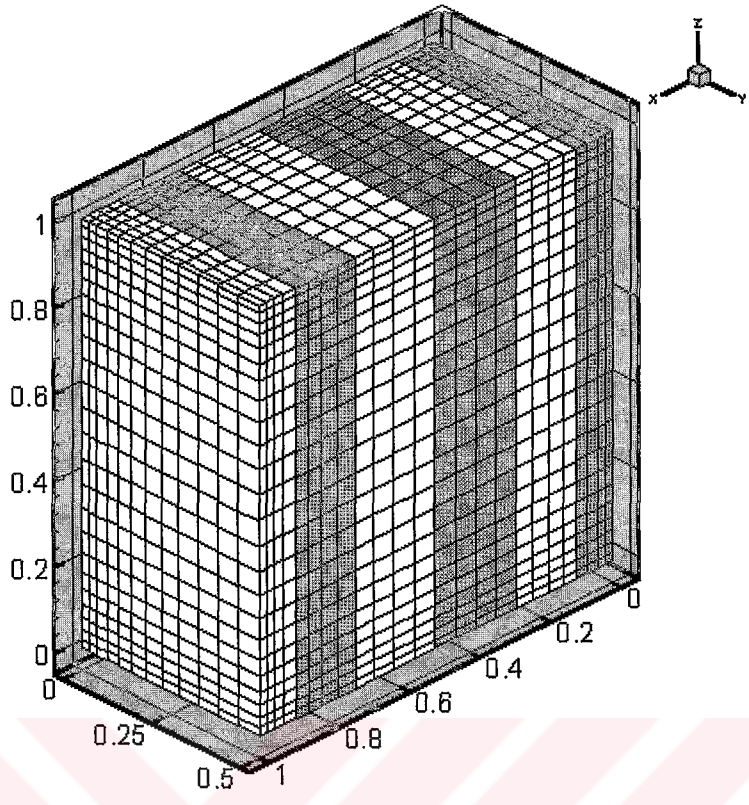


Figure 7.4. Cavity grid, 6-domain partitioning, 6x(5x13x25) nodes

7.2. First Parallelization Results and Discussion

The corrections made in the implicit code, noted in the previous chapter, have been carried to the parallel code written in this thesis. The results have been obtained for 2, 4 and 6 sub-domains on DEC Alpha XL266 workstations. The problem in running the computational code for the cubic cavity problem on 6-domain partitioning is solved after some difficulties; the reason of the error was the dimension opened for the arrays in domain decomposition method. As it is seen in the flow chart of the first parallel implementation in Chapter 5, the master processor collects the interface velocity and auxiliary potential from each slave. When the interface number is increased, the total number of transmitted unknowns is increased. Because the dimension opened in the master processor was not big enough to collect the 6 interface values, they were not being collected properly, so causing the problem in 6-domain partitioning case.

Figure 7.5, Figure 7.6 and Figure 7.7 give the pressure contour plots on the symmetry-plane at steady state, for 2, 4 and 6-domain partitioning, respectively. Pressure values on different contours can be read from these figures. Pressure distribution is not oscillatory inside the cavity. Near the lid, little oscillations are observed, because the auxiliary potential function has to be fixed to zero along the lid as a boundary condition, which is in fact not physical.

The velocity vector field at steady state on the symmetry plane for 2-domain partitioning is presented in Figure 7.8, together with constant x plane plots on different cross-sections of the cavity. Moving lid drives a re-circulating flow inside the cubic cavity, convection is dominant and the primary vortex is offset towards the geometric center of the cavity. At the lower corner, a second vortex on the symmetry plane is visible. The velocity vector field for 4-domain partitioning run is represented by Figure 7.9, together with the plots on different constant x planes, which are perpendicular to the lid. Secondary flows along the top and bottom edges of the cubic cavity can be observed. Figure 7.10 shows the plots of velocity vector field on the symmetry-plane and on different constant z planes for 6-domain partitioning.

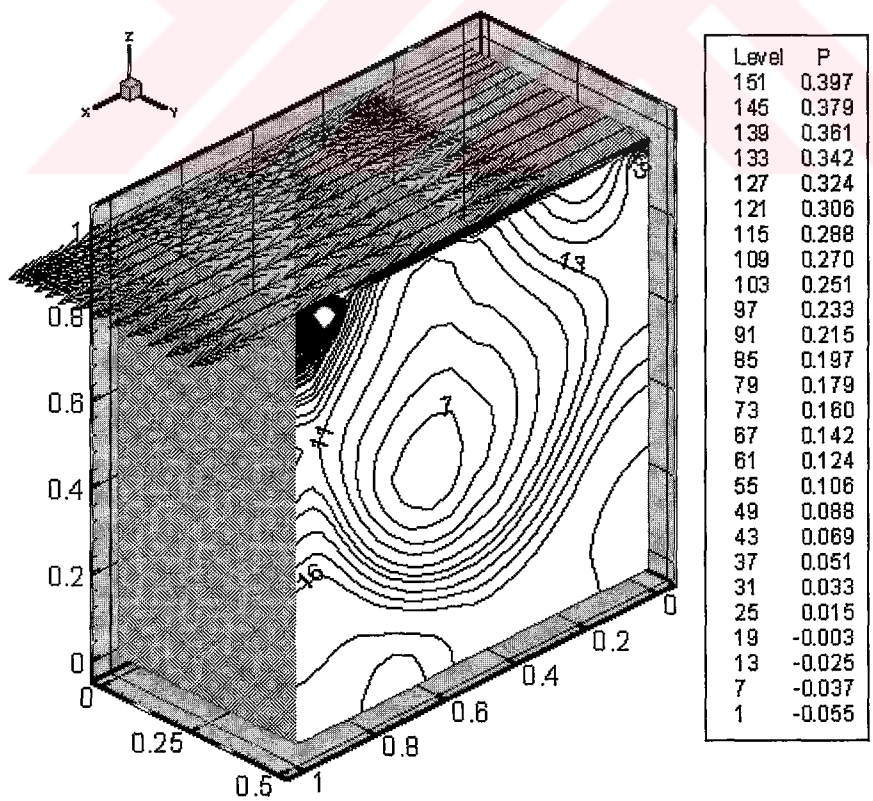


Figure 7.5. Symmetry plane pressure contours, first parallelization result with 2-domain partitioning, 2 x (13x13x25), Re=1000, Δt=0.1.

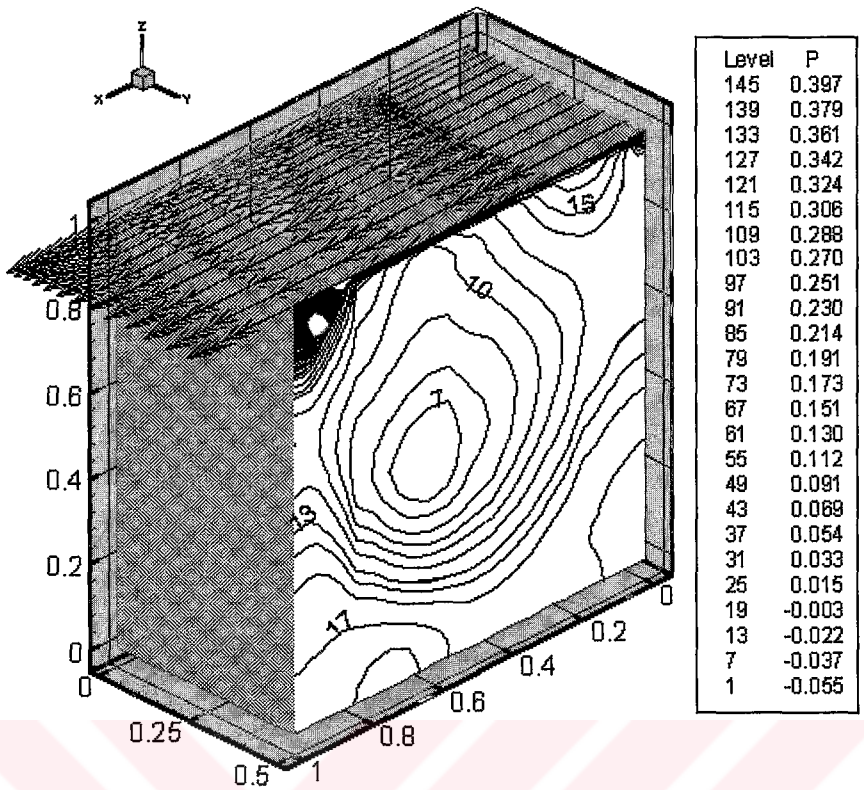


Figure 7.6. Symmetry plane pressure contours, first parallelization result with 4-domain partitioning, 4 x (7x13x25), Re=1000, Δt=0.1.

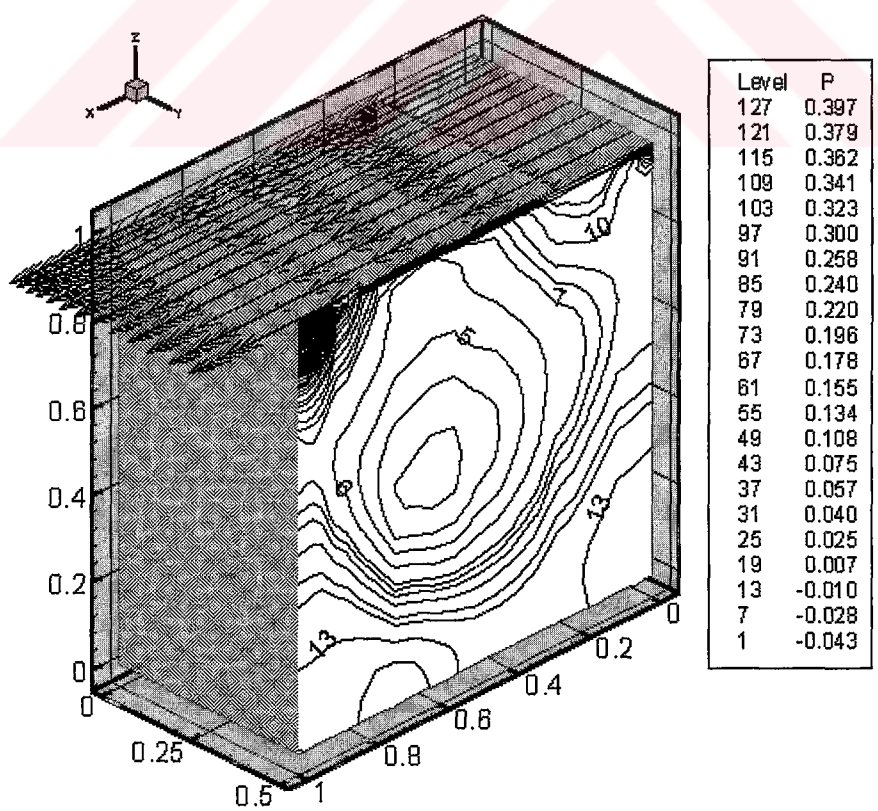
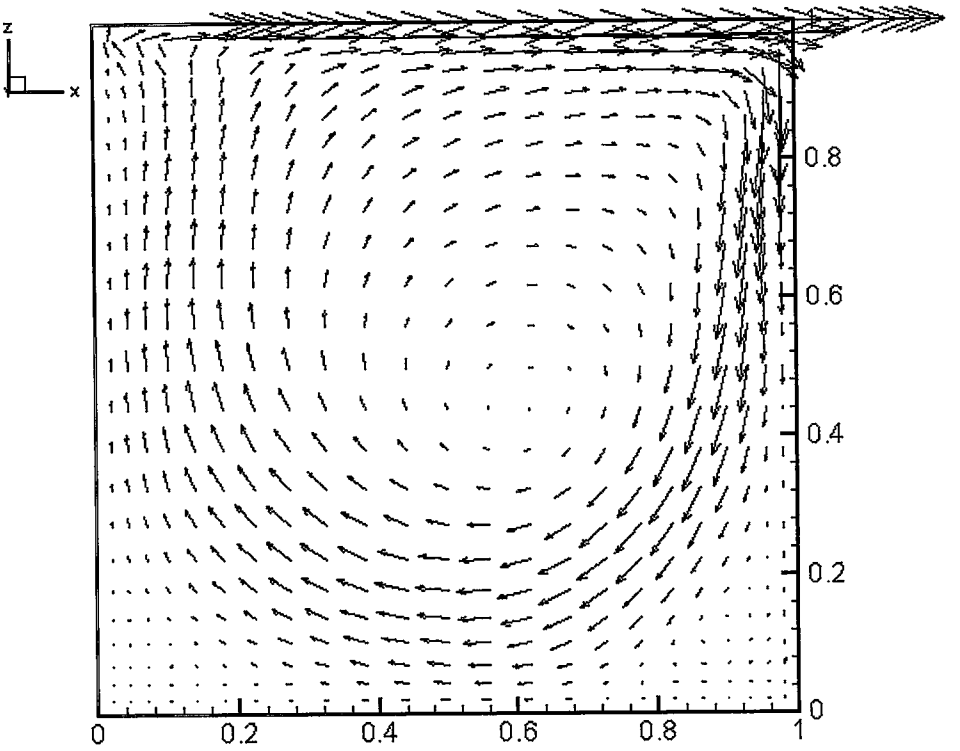
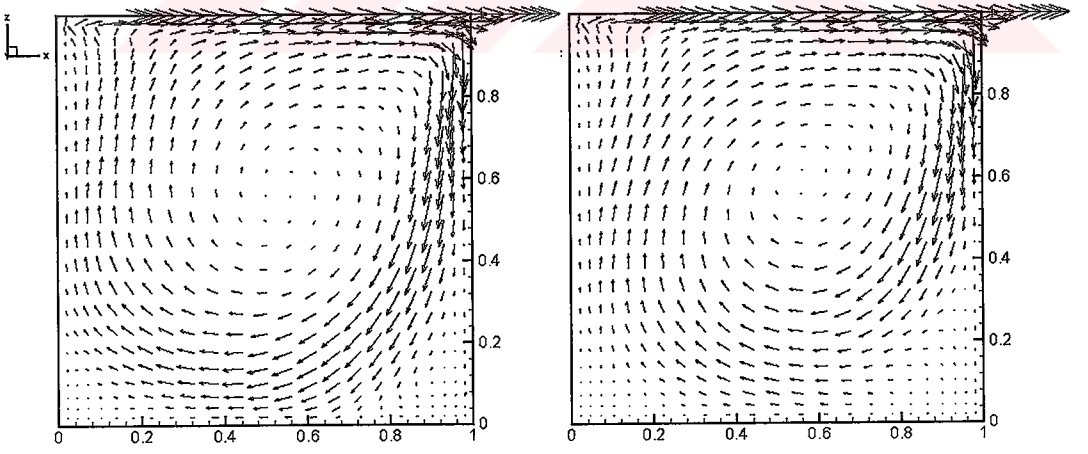


Figure 7.7. Symmetry plane pressure contours, first parallelization result with 6-domain partitioning, 6 x (5x13x25), Re=1000, Δt=0.1.



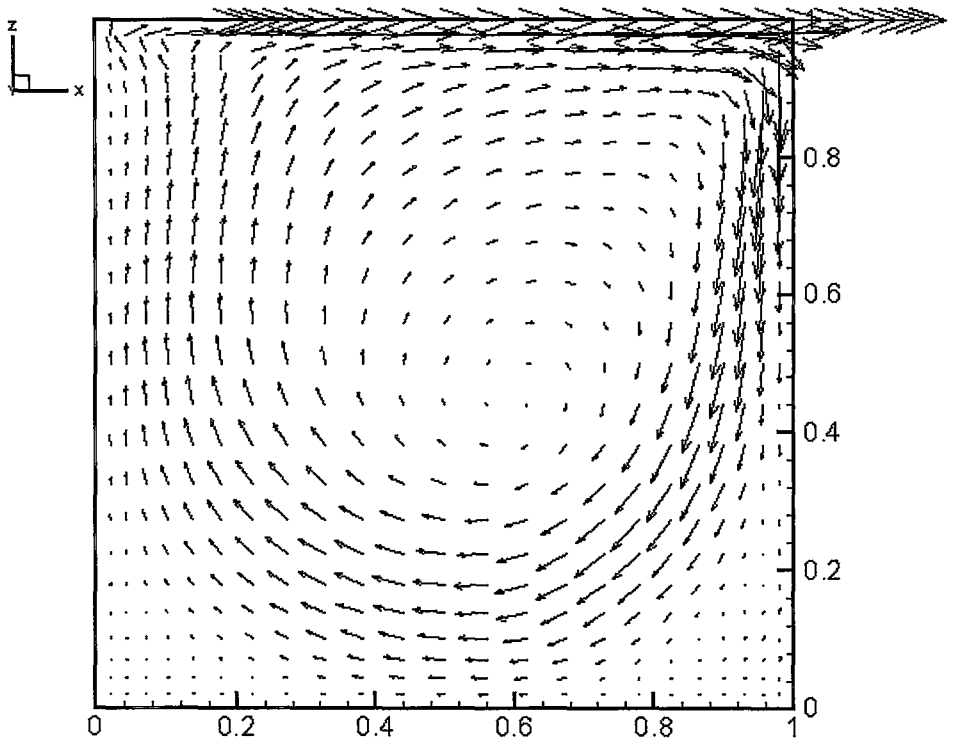
(a)



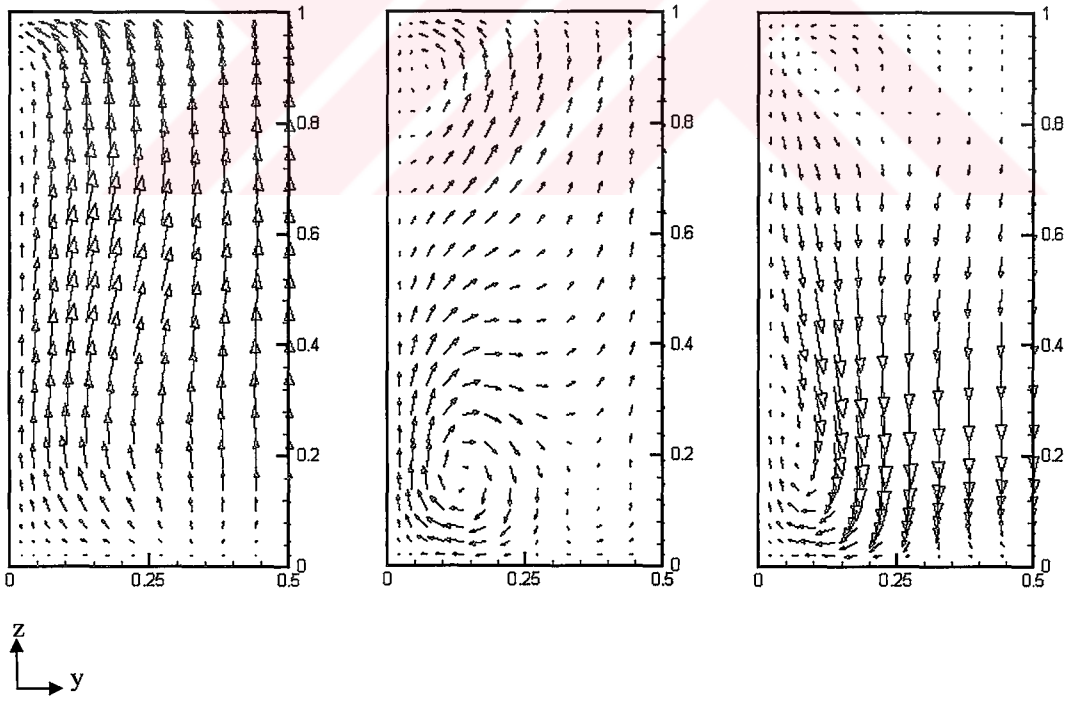
(b)

(c)

Figure 7.8. Velocity vector field, first parallelization results, on 2-domain partitioning 2 x (13x13x25): (a) y = 0.5 (symmetry-plane) (b) y = 0.25 (c) y = 0.1 plane, Re=1000, Δt=0.1.

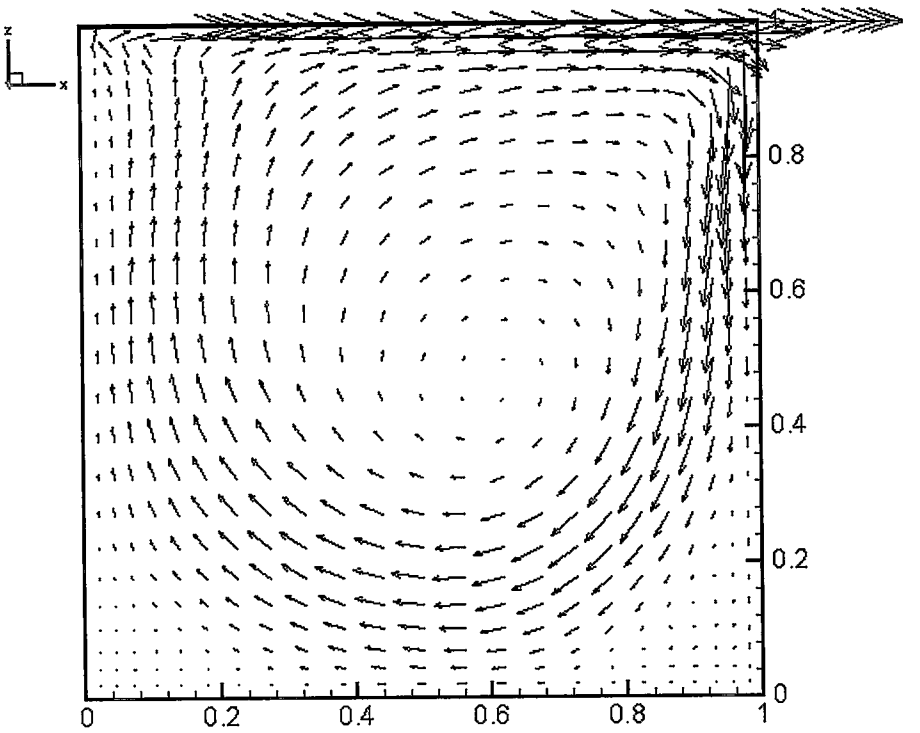


(a)

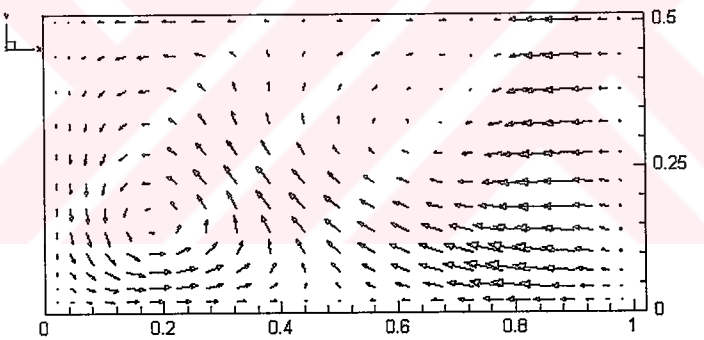


(b) (c) (d)

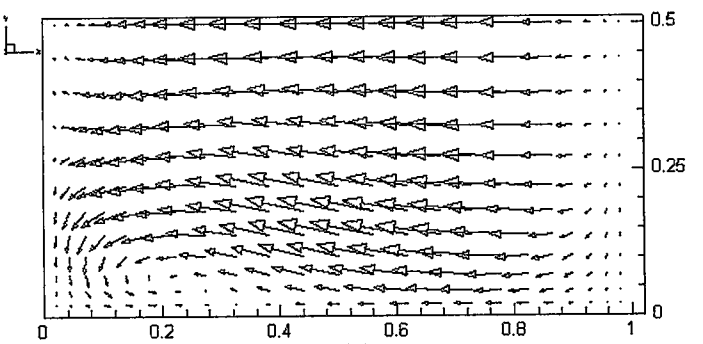
Figure 7.9. Velocity vector field, first parallelization results, on 4-domain partitioning, 4 x (7x13x25): (a) y = 0.5 (symmetry-plane) (b) x = 0.25 (c) x = 0.5 (d) x = 0.75 plane, Re=1000, Δt=0.1.



(a)



(b)



(c)

Figure 7.10. Velocity vector field, first parallelization results, on 6-domain partitioning 6 x (5x13x25): (a) y = 0.5 (symmetry-plane) (b) z = 0.5 (c) z = 0.25 plane, Re=1000, Δt=0.1.

The plots show that interface solutions are successful and Neumann condition on the interfaces is satisfied correctly with the domain decomposition algorithm in Chapter 5. Parallel solutions agree well with the sequential results, accuracy is not affected with the parallelization. The advantage is gained in terms of computational time, and working on large size problems becomes possible with parallel computation. The solution is advanced up to the dimensionless time level of 25, where the steady state is reached with a large time-step size of 0.1. This is one of the big advantages of the parallel implicit code that, there is no restriction on time-step size as in the explicit methods.

Horizontal velocity at vertical centerline and vertical velocity at horizontal centerline, on the symmetry-plane at steady state, is plotted in Figure 7.11 for the first parallelization. The results obtained with 2, 4 and 6 sub-domains are compared with the benchmark solution in reference Ku *et.al.*, [71]. They give reasonably well agreement with the benchmark solutions, although the grid is much coarser than the one used in the benchmark solution, and the stretching near the walls is very loose.

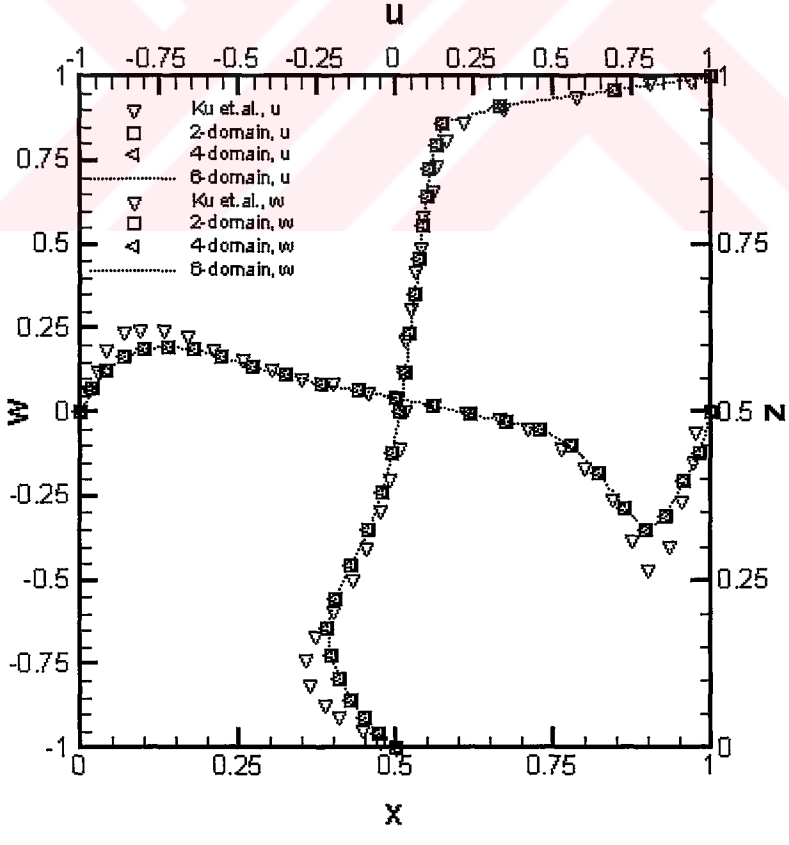


Figure 7.11. Velocity distributions along the horizontal & vertical centerlines of the symmetry-plane of lid-driven cubic cavity flow, 2, 4 and 6-domain partitioning results, first parallelization, Re=1000, Δt=0.1.

With the first parallelization run with 2-domain partitioning, the interface inner iterations for the half-time step velocity at each time step stay generally at 7 iterations, and auxiliary potential interface iterations are reached up to 55 iterations. Half-time step velocity interface iterations generally count 8 and 9 for 4-domain and 6-domain partitioning runs, respectively. Auxiliary potential interface iterations reach up to 85 and 105 for 4-domain and 6-domain partitioning runs, respectively.

Although the velocity interface iterations number is very small, auxiliary potential one is a lot. This means that, the program spends a long time to be able to find the optimum condition for the auxiliary potential at the interfaces at each time-step. Since the interface iteration number increases and the number of communications between master and slaves increases, the computational time increases and speed-up decreases. During this study, researches are also concentrated at this point and numerical experiments are done to find a way to decrease the auxiliary potential function interface iteration number.

As the number of interface increases, iteration number also increases, as expected. Master makes scaling, based on the total number of interface nodes, while calculating the difference between the old and new iteration level values of unknowns. The master processor does this scaling, before making the convergence check as explained in detail in Chapter 5.

The computational times for 1, 2, 4 and 6-domain partitioning obtained on DEC Alpha XL266 workstations are compared in Table 7.1. These results show that there is a problem with the first parallelization. Based on the 2-domain partitioning solution, the 4-domain solution speed-up is 1.37 as opposed to its ideal value, 2, and 6-domain solution speed-up is 0.89.

Speed-up analysis is continued in the following sections in detail, and the problem with the low speed-up value, obtained with the first parallelization, is overcome. Especially, the master processor causes this problem, the computational time of the master processor solution increases, as the number of sub-domains and of interface increases. Load balance is going to be done in the following parallelization ways and the speed-up is going to be increased on.

Table 7.1. Computational times of 1, 2, 4 and 6-domain partitioning, the first parallelization results, on DEC Alpha XL266 workstations.

GRID	CPU(s) Master	CPU(s) Slave 1	CPU(s) Slave 2	CPU(s) Slave 3	CPU(s) Slave 4	CPU(s) Slave 5	CPU(s) Slave 6	ELAPSED (s)
1x (5x13x25)	13369	.....	.....	.....	.....	.....	.....	21211
2x (13x13x25)	531	4000	3884	.....	.....	.....	.....	4448
4x (7x13x25)	1755	1260	1500	1472	1107	.....	.....	3229
6x (5x13x25)	3423	813	1051	1063	648	1075	805	4979

7.3. Second Parallelization Results and Discussion

After having the solutions on 2, 4 and 6 domains, the results showed us that there is a problem in efficiency as the sub-domain and interface number increases. The elapsed time for 6-domain solution is increased to 4979 seconds, which is bigger than the value of elapsed time taken for 4 domain and even 2 domain solutions. The efficiency is so poor. Although the CPU times for the slave processors are decreased as shown in Table 3, there was a problem with master CPU time. Higher the interface number, longer the master CPU time. The interface values to be collected

and processed by the master for domain decomposition method are increased at each interface iteration cycle as shown in the flow chart in Chapter 5 in Figure 5.3.

The computer used in this work as a master processor is not as fast as the other 6 slave processors. Master is responsible for post-processing, all interface values are processed by the master as explained in Chapter 5 and the slaves cannot go on the calculations without having the results from the master. So quality and speed of master processor are important factors on efficiency of the parallel computations. At each time-step, the slaves are waiting too much for the master to complete its work, described in the flow chart (Figure 5.3), to be able to go on their calculations.

Behind these problems, the data to be processed by the master increase as the interface number increases and this causes much more increase in computation time of the master calculations. If the master processor is slow as in this study, overall computational time increases too much, especially in 6-domain partitioning run. All of these problems cause the poor efficiency in speed-up analysis.

So a new method of parallelization is investigated to make a load balance (considering the low capacity of master processor) and to get more efficient parallel results on DEC Alpha XL266 workstations.

The principle of this second implementation is to give the master processor less work and to decrease the amount of arrays to be transmitted, of communications between the slave and master processors and of the data to be post-processed by the master, at each inner interface iterative cycle of the domain decomposition method. In this second implementation, the master processor only collects the interface arrays obtained at slaves for the steepest descent part described in Chapter 5; it calculates only the scalar coefficients defined at steepest descent part, sends them to the slaves and decides the convergence at each inner iterative cycle.

On the other hand, communications between the slave processors exist in this parallelization, opposite to the first parallelization method. Each slave communicates with neighbouring one. The slave processors, which have the send interface boundary, do the steepest descent calculations instead of master, at each inner iterative cycle. The difference between the old and new iteration values of the

velocities and the pressure at the interfaces is found at the slaves. Then each slave sends the difference to the master for only convergence check at each cycle as described in Chapter 5. The flow chart of second parallel implementation is shown in Figure 7.12.

To load the most work of domain decomposition to the slave processors instead of the master processor decreases the elapsed time for each 2, 4 and 6-domain computations on DEC Alpha XL266 workstations, for the cubic lid-driven cavity problem test case. The accuracy of the results is not changed, but still the efficiency in parallel method needs to be improved. Master CPU time still increases as interface number increases, so causing the increase in elapsed time. The analysis is shown in Table 7.2 – 7.7 and in Figure 7.14. This brought us to further developments in parallel computations on DEC Alpha XL266 workstations, as follows.

7.4. Third Parallelization Results and Discussion

It is seen that if the master processor does the work at each interface inner iteration cycle, communicates with all slaves and too much data is transported between the machines, the elapsed time increases and so the efficiency is poor. This result brought us to a new idea of parallelization way. The load balance is done in such a way that, the slave processors handle almost all of the domain decomposition calculations, so the decrease in parallel efficiency, caused by the slow master processor, can be prevented.

The principle of the third parallel method is to set the master processor out of the inner iteration cycle at each time step and to load all the domain decomposition process on the slaves. In this method master only spawns the slaves and finishes its task.

The slaves solve the initialization part described in Chapter 5. Initial half-time step velocity field and auxiliary potential function values are obtained at each interface. Then the slave processors, which have the send interface boundary nodes, collect these values from the neighboring one and calculate the differences at their interfaces before starting the unit problem solution part and the steepest descent calculations,

defined in Chapter 5. The arrays to be transported are decreased and most of the communication is made between the neighboring slaves.

The last slave only sends its interface values and does not have to make the steepest descent calculations. So the calculation of the scalar coefficients at the steepest descent part defined in Chapter 5, and the convergence check for inner iteration cycle at each time step tasks, are given to this last slave, which will play the role of the slow master processor in this study. After finding the correct Neumann boundary conditions on the domain interfaces, each slave, which has the send interface boundary nodes, sends the converged interface values to the neighboring one and finalization part starts at each sub-domain, as explained in Chapter 5.

The most efficient speed-up is obtained when the last slave is set on vm06 machine (see Figure 5.1a), which was observed to be faster than the other machines used in the parallel processing. Since it is fast, the other slaves do not wait too much for the last slave to send the scalar coefficients at steepest descent part and convergence check result at each inner interface iteration cycle.

The computational code modified again with this third parallel implementation method is calibrated with the cubic lid-driven cavity problem with Reynolds number of 1000. For 2-domain partitioning, 2x(13x13x25) nodes, for 4-domain partitioning 4x(7x13x25) nodes, and for 6-domain partitioning 6x(5x13x25) nodes are used as shown in Figure 7.2–7.4. The steady state is reached with a large time-step size of 0.1, at 25 dimensionless-time.

Part of flow chart concerning the parent (master) and the child (slave) processes for this latest parallel implementation is shown in Figure 7.13. Overall computational time (elapsed time) for 2, 4 and 6-domain partitioning runs using the third parallelization way on DEC Alpha XL266 workstations are written in Table 7.2 – 7.7 and in Figure 7.14. The comparison, in terms of computational time, speed-up, and parallel efficiency, of the three methods of parallelization described up to now can be followed from Table 7.2 – 7.7 and from Figure 7.14, too.

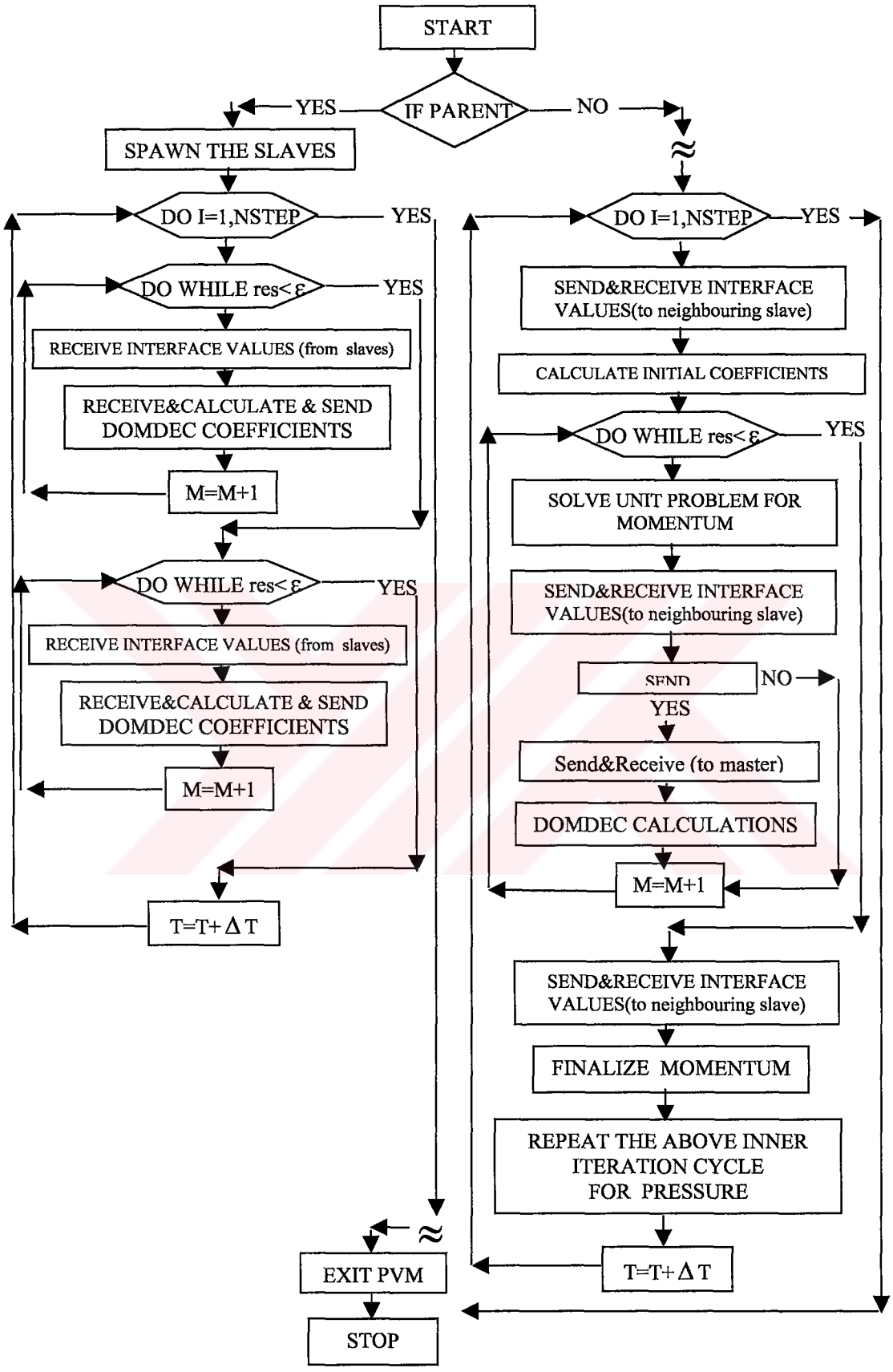


Figure 7.12. Flow chart of the second parallel implementation, concerning the parent and child processors.

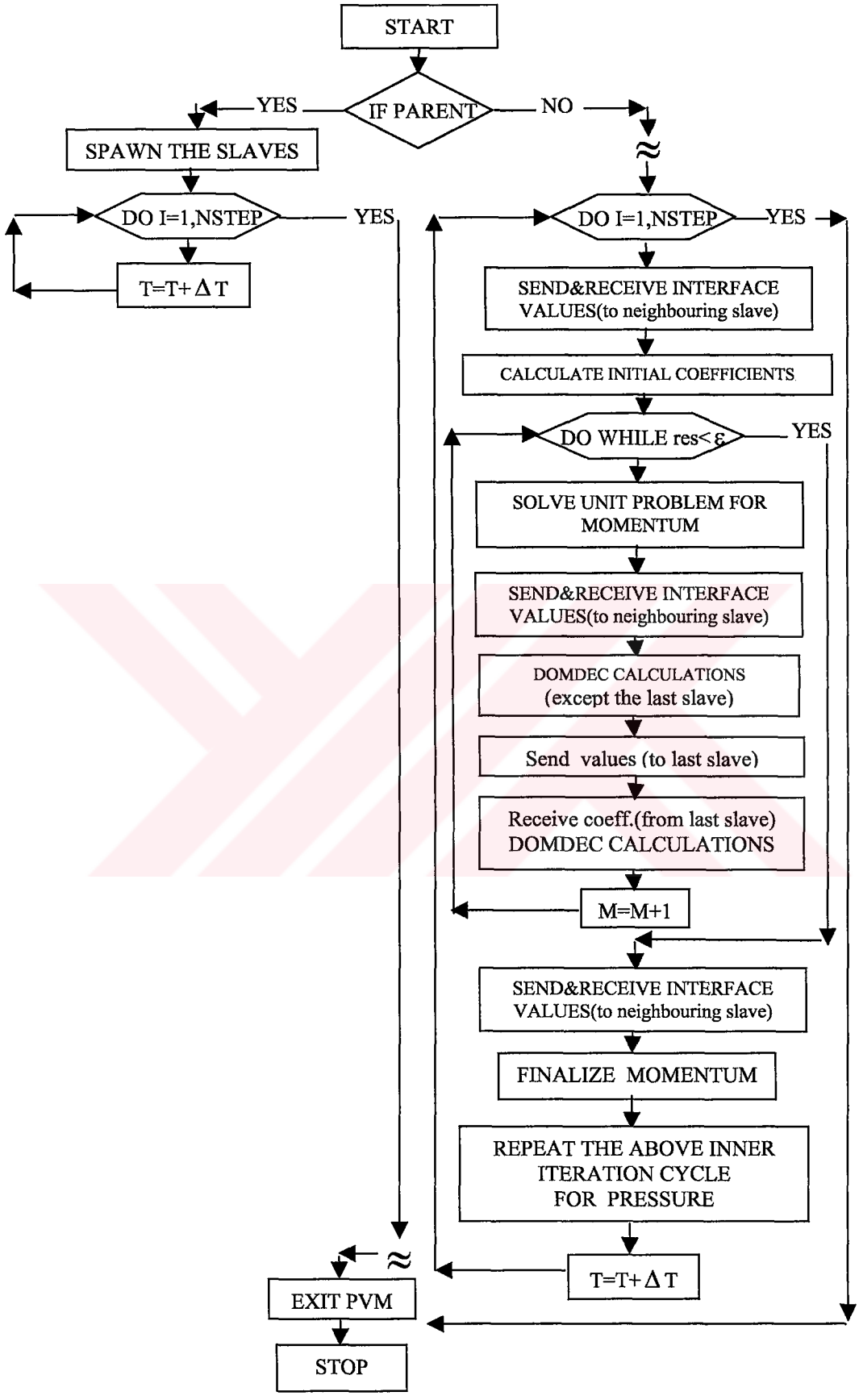


Figure 7.13. Flow chart of the third parallel implementation, concerning the parent and child processors.

As indicated in section 7.2, the speed-up of the 4-domain solution obtained with the first parallelization, on Alpha XL266 workstations, is 1.37 as opposed to its ideal value, 2. With this third improvement, the elapsed times for 2, 4, and 6-domain partitioning solutions of the three dimensional lid-driven cavity problem are decreased significantly. Load balance studies in the parallel program run on the workstations removed the problem caused by the slow master processor and improved the efficiency of the parallelization method. The speed-up concerning the elapsed time is increased to 2.39 for 4-domain partitioning and to 2.60 for 6-domain partitioning runs, where the normalization is done with respect to the elapsed time of 2-domain partitioning computation.

The amount of the decrease of elapsed time in 4-domain solution is much more than that of 6-domain case; this is probably related to the size of half-bandwidth difference between the partitioning cases. The larger the size of half-bandwidth, the more amount of data to be processed and transported. As indicated before, the half-bandwidths for 2-domain, 4-domain and 6-domain solutions are 184, 100 and 72, respectively, where much more decrease of half-bandwidth is obtained in 4-domain partitioning case compared to that of the 6-domain case.

Table 7.2. Comparison of elapsed time for the first, second and third parallelization computations for cubic lid-driven cavity flow, Re=1000, Δt=0.1, DEC Alpha XL266 workstations.

25x13x25 total grid points	CUBIC CAVITY (Re=1000, Δt=0.1) ELAPSED TIME (s)		
Number of domains	2	4	6
1st parallel implementation	4448	3229	4979
2nd parallel implementation	4402	3068	4046
3rd parallel implementation	4105	1719	2190
3rd parallel implementation (Last slave is on vm06)	4105	1719	1579
Explicit Method, [51], 25x25x25 total grid points	6.32 hour	2.92 hour	

Table 7.3. Comparison of master (parent) processor CPU time for the first, second and third parallelization computations for cubic lid-driven cavity flow, Re=1000, Δt=0.1, DEC Alpha XL266 workstations.

25x13x25 total grid points	CUBIC CAVITY (Re=1000, Δt=0.1) MASTER CPU TIME (s)		
Number of domains	2	4	6
1st parallel implementation	531	1755	3423
2nd parallel implementation	427	1312	1952
3rd parallel implementation	0.576	0.498	0.322
3rd parallel implementation (Last slave is on vm06)	0.576	0.498	0.322

Table 7.4. Comparison of slave (child) processors CPU time for the first, second and third parallelization computations with 2-domain partitioning case, Re=1000, Δt=0.1, DEC Alpha XL266 workstations.

25x13x25 total grid points	CUBIC CAVITY, 2-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)	
Slave order	1st	2nd
1st parallel implementation	4000	3884
2nd parallel implementation	4182	4004
3rd parallel implementation	3989	3981

Table 7.5. Comparison of slave (child) processors CPU time for the first, second and third parallelization computations with 4-domain partitioning case, Re=1000, Δt=0.1, DEC Alpha XL266 workstations.

25x13x25 total grid points	CUBIC CAVITY, 4-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)			
Slave order	1st	2nd	3rd	4th
1st parallel implementation	1260	1500	1472	1107
2nd parallel implementation	1764	1943	1869	1096
3rd parallel implementation	1511	1511	1509	1205

Table 7.6. Comparison of slave (child) processors CPU time for the first, second and third parallelization computations with 6-domain partitioning case, Re=1000, Δt=0.1, DEC Alpha XL266 workstations.

	CUBIC CAVITY, 6-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)					
Slave order	1st	2nd	3rd	4th	5th	6th
1st parallel implementation	813	1051	1063	648	1075	805
2nd parallel implementation	1453	1659	1602	752	1494	821
3rd parallel implementation	1482	1635	1599	732	1445	1783
3rd parallel implementation (Last slave is on vm06)	1333	1293	1312	1329	1280	848

Speed-up values are shown in Table 7.7 for each of the parallelization methods. When compared to the explicit result of 4-domain speed-up (elapsed) of 2.17, obtained in reference [51] on the same DEC Alpha XL266 workstations, our implicit parallel solution speed-up (elapsed) reaches to 2.39 with 4-domain computation and 2.60 with 6-domain computation. In Reference [5], cubic cavity flow with Re=1000 is computed in 6.32 h. with 2-domain partitioning solution and in 2.92 h. for 4-domain partitioning with 25x25x25 equally spaced grid points. The convergence is obtained after 625 time steps up to the dimensionless time level of 25. The results shown in this chapter are obtained in a shorter time than the explicit reference one, [51]. The advantage is ability to take large time-step size of 0.1. Our results are converged in 250 time steps up to dimensionless time level of 25 for 2, 4 and 6-domain partitioning with 2x(13x13x25), 4x(7x13x25) and 6x(5x13x25) grid points, respectively.

Figure 7.14 shows clearly the parallel efficiency of the third parallelization method, relative slave CPU times are plotted for each slave at 2, 4, and 6-domain partitioning runs for cubic lid-driven cavity problem.

Figure 7.15 compares the velocity distribution results, at horizontal and vertical centerlines of the symmetry plane of the cavity, obtained with the third parallelization method. 6-domain partitioning results are compared with the 6-

domain results of the first parallelization method, accuracy is not affected with the modifications in parallelization way. The number of interface iterations for half-time velocity and auxiliary potential does not change, but significant improvement is succeeded in terms of efficiency.

Table 7.7. Comparison of speed-up (elapsed time) results for the first, second and third parallelization computations for cubic lid-driven cavity flow, Re=1000, Δt=0.1, DEC Alpha XL266 workstations.

	CUBIC CAVITY (Re=1000, Δt=0.1)		
Number of domains	2	4	6
Speed-up (elapsed time) First parallelization	1	1.37	0.89
Speed-up (elapsed time) Second parallelization	1	1.43	1.09
Speed-up (elapsed time) Third parallelization	1	2.39	2.60
Speed-up (elapsed time) Explicit Method, [51]	1	2.17	

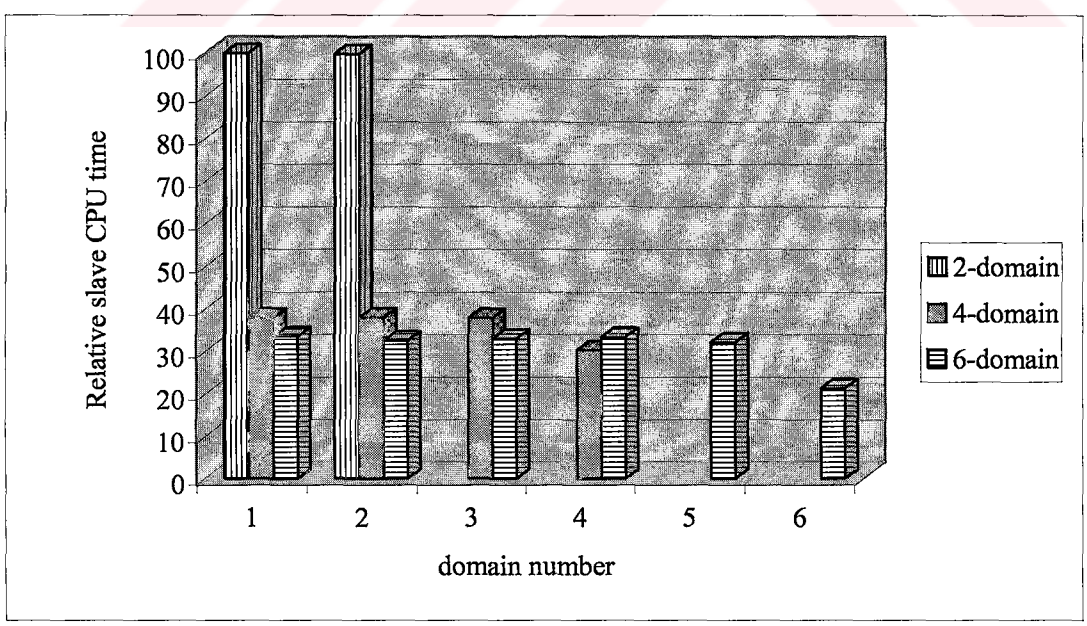


Figure 7.14. Third parallelization result: relative CPU time of each processor, 2, 4 and 6-domain partitioning, Re=1000, Δt=0.1, DEC Alpha XL266 workstations.

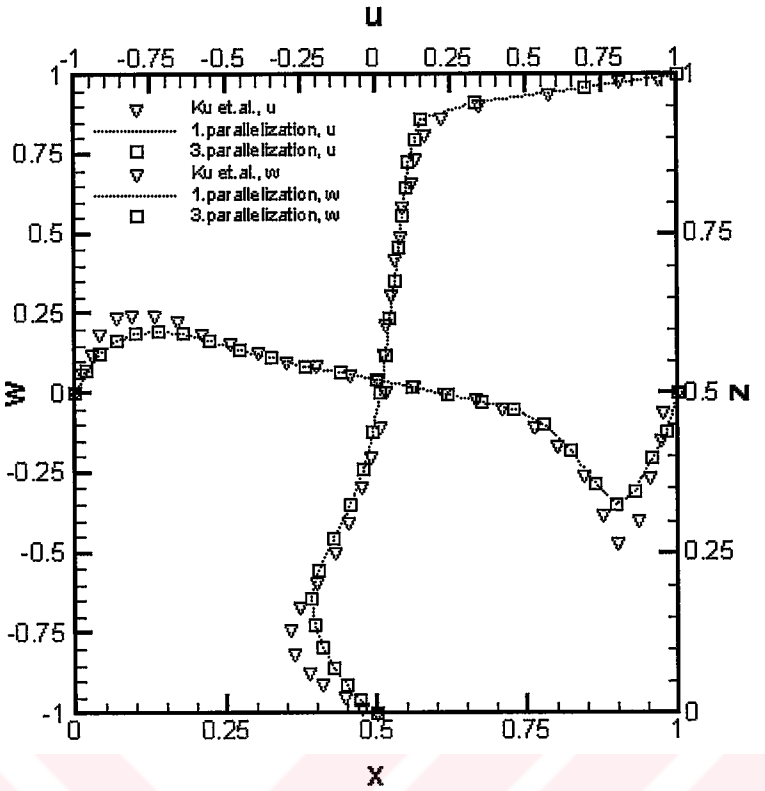


Figure 7.15. Comparison of first and third parallelization, velocity distribution at horizontal & vertical centerlines of the symmetry plane, 6-domain partitioning case, 6 x (5x13x25) nodes, Re=1000, Δt=0.1, DEC Alpha XL266 workstations

To obtain high parallel efficiency;

i) If all the processors used in the parallel computations have the same quality, speed and communication speed, all the work to solve the interested problem should be distributed equally to each processor,

ii) The speed of the master processor, which is responsible for the post-processing of all the interface data, is the important factor on parallel computation time and speed-up. Especially if the interface number is high; the more the interface, the more the data to be processed and the more the communications between the master and the slaves,

iii) If the processors used in the parallel computation are not equal, load balance should be done in such a way that, the most of the work is loaded to the fast processors, processors should not have to wait for another one to continue their own computations.

7.5. Large Size Problem: lid-driven cubic cavity flow at Re= 1000

The parallel implementation problems are solved as discussed in the previous chapters. Ideal speed-up is reached with the computations on DEC Alpha XL266 workstations. After the latest developments, it is aimed to run the latest parallel implicit program (third parallelization) to compute a large size problem.

But there is a problem in running the code with the large size grid. The size of the matrices to be inverted enlarges; therefore size of the code increases too much to run on the processors used in this study. So the code required high memory. One of the large size matrices is the assembled stiffness matrix **A** defined in Chapter 3 (equation 3.20). It is first being assembled, factored and finally stored on disk at the first time step but not needed later on. So instead of keeping this matrix, it is decided to write the factored one on a data file once and then read this data during computations explicitly. So we could get rid of this huge size, number of nodes x half-bandwidth, matrix.

The other large size matrix is the coefficient matrix of the half-time step velocity, $\left(\frac{2\mathbf{M}}{\Delta t}+\mathbf{D}+\frac{\mathbf{A}}{\mathrm{Re}}\right)$, at equation (3.17) in Chapter 3. As the assembled stiffness matrix, it is also being factored and stored on the disk with a large dimension, number of nodes x (2xhalf-bandwidth-1). It is needed at each time step, so cannot be removed. But instead of keeping both the assembled form of this matrix and the vector form of its upper and lower triangular matrices, it is decided to compute directly the assembled symmetric arrays of that matrix, without constructing it. With these eliminations, the size of code is reduced significantly and became ready to run on the existing processors.

Large size lid-driven cubic cavity test domain is used to calibrate the parallel implicit code. Reynolds number of 1000 is chosen. 6-domain partitioning computation data is prepared with 6x(9x25x49) grid points. Both uniform and non-uniform grids shown in Figure 7.16 and Figure 7.17, respectively, are used in the computations.

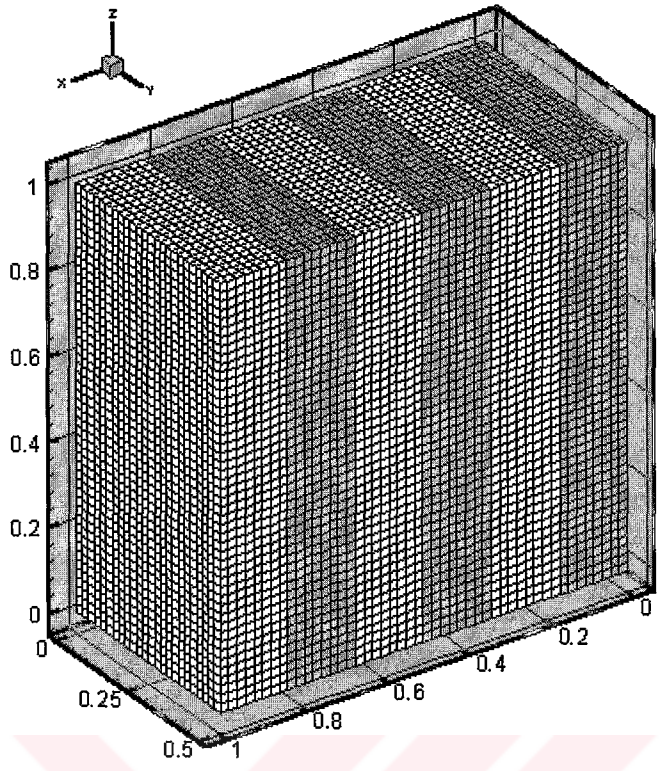


Figure 7.16. Uniform 6-domain partitioning grid, 6x(9x25x49) nodes

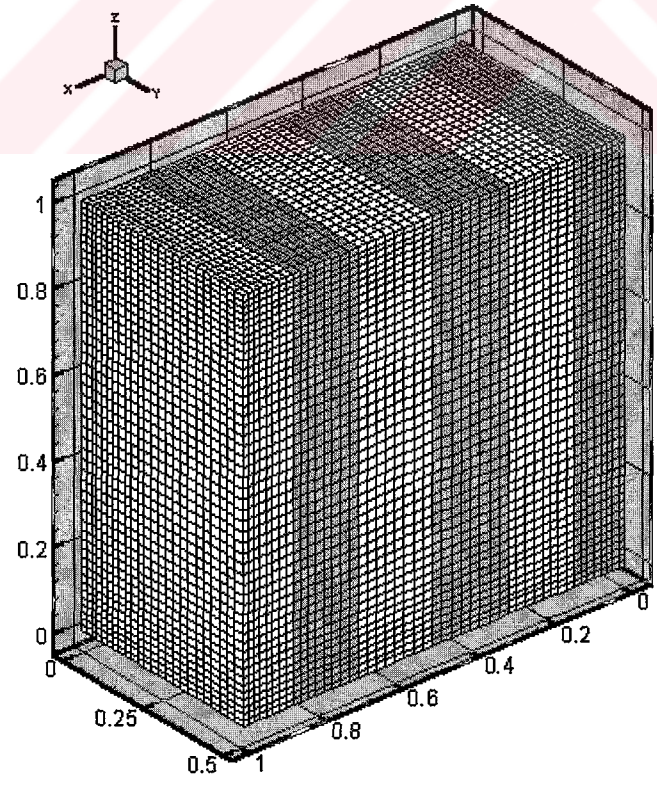


Figure 7.17. Non-uniform 6-domain partitioning grid, 6x(9x25x49) nodes

The steady state is reached after 500 dimensionless time, with a time-step size of 0.05. The finest grid spacings of 6x(9x25x49) uniform and non-uniform grids are 0.02(1:50) and 0.0142 (1:70), respectively.

The comparison of horizontal and vertical velocities at the centerlines of the symmetry plane of the cubic cavity can be seen in Figure 7.18, including both uniform and non-uniform grids. With the non-uniform grid with a 1:70 finest grid spacing (stretched near the side walls), the results agree better, than the uniform grid result, with those of benchmark solutions of Ku *et al.*, [71], who uses pseudo-spectral method in their computations with finer grids, on a Cray. So the results accuracy could be improved more, especially with finer mesh near the sidewalls, as seen in figures 7.20 –7.23. The pressure contours on the symmetry plane, for both uniform and non-uniform grids, are shown in Figure 7.19. Pressure field obtained with uniform grid is not as stable as the one obtained with the non-uniform grid. The non-uniform grid result is much better than the uniform one.

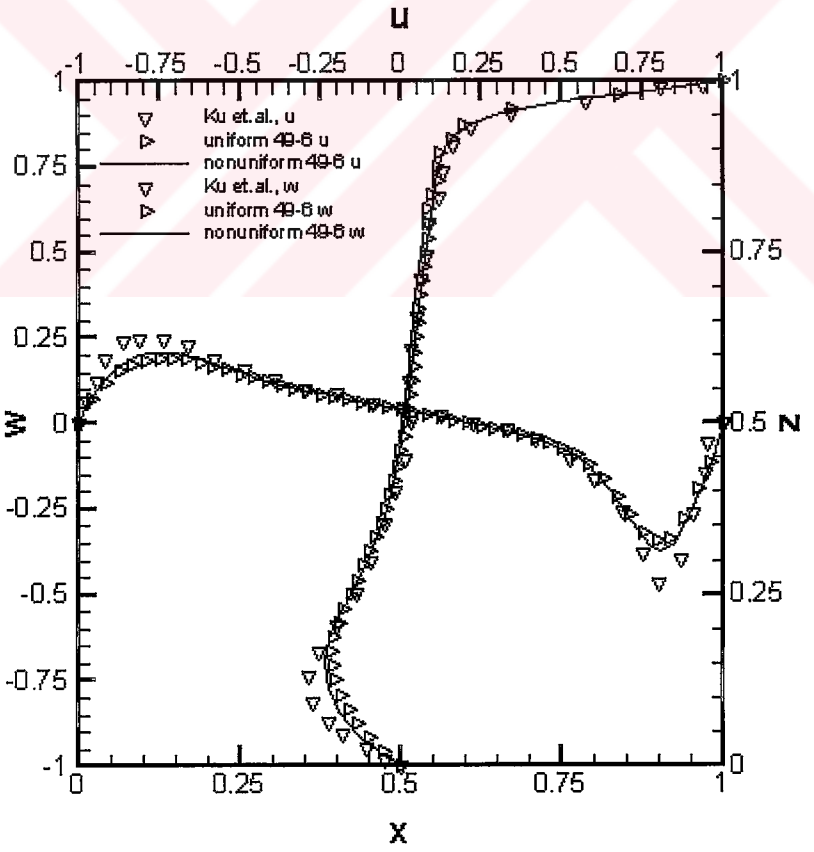


Figure 7.18. Comparison of uniform and non-uniform grid, velocity distributions along the centerlines of the symmetry-plane, 6-domain partitioning, third parallelization, Re=1000.

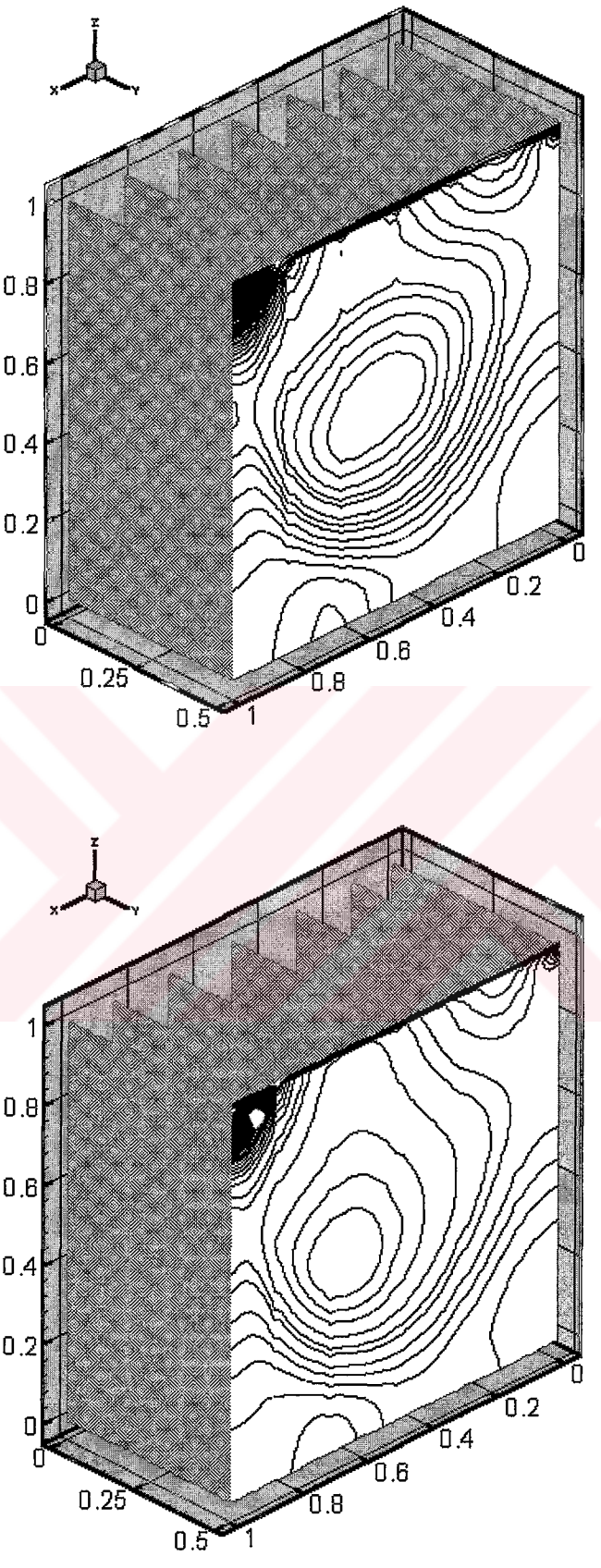


Figure 7.19. Uniform (top figure) and non-uniform grid results, symmetry plane pressure contours, third parallelization results, 6-domain partitioning with 6 x (9x25x49) nodes, Re=1000, DEC Alpha XL266 workstations.

Elapsed time of third parallelization method computation with the non-uniform 6x(9x25x49) grid is found out as 38988 seconds on DEC Alpha XL266 workstations. On the other hand, it is found out as 49533 seconds, if the first parallelization way is employed; it is about 10000 seconds longer.

To see the affect of fine grid on the accuracy of the results, especially the stretching affect, the finer grid, with total number of 6 x (10x28x55) nodes and with finest grid spacing of 1:77, is used in 6-domain partitioning run. Contour plot can be seen in Figure 7.20 and the velocity vector fields on the symmetry plane and on different cross-sections of the cavity are plotted in Figure 7.21.

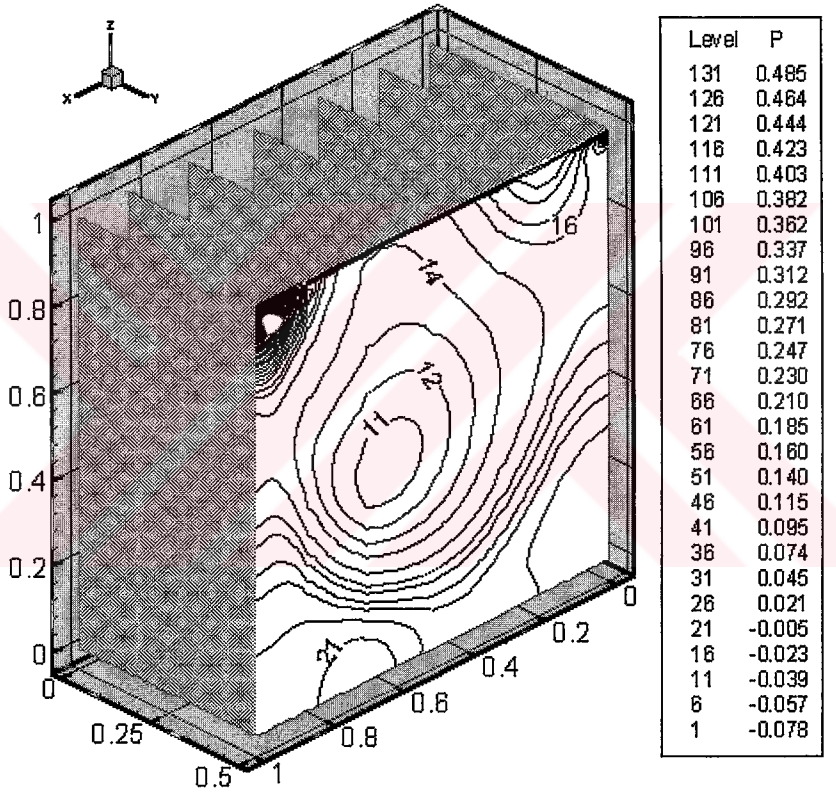


Figure 7.20. 6-domain partitioning 6 x (10x28x55) result, third parallelization, symmetry-plane pressure contours of the lid-driven cavity flow, Re=1000, on DEC Alpha XL266 workstations

The best convergence criteria for the interface inner iterations are found out as $\varepsilon_{\mathbf{u}}=10^{-5}$ for half-time velocity field and $\varepsilon_{\phi}=10^{-6}$ for the auxiliary potential function.

The half-time velocity and auxiliary potential interface iterations reach 7 and 79 iterations, respectively, with the third parallelization computation for 6-domain partitioning of total number of 6 x (10x28x55) nodes.

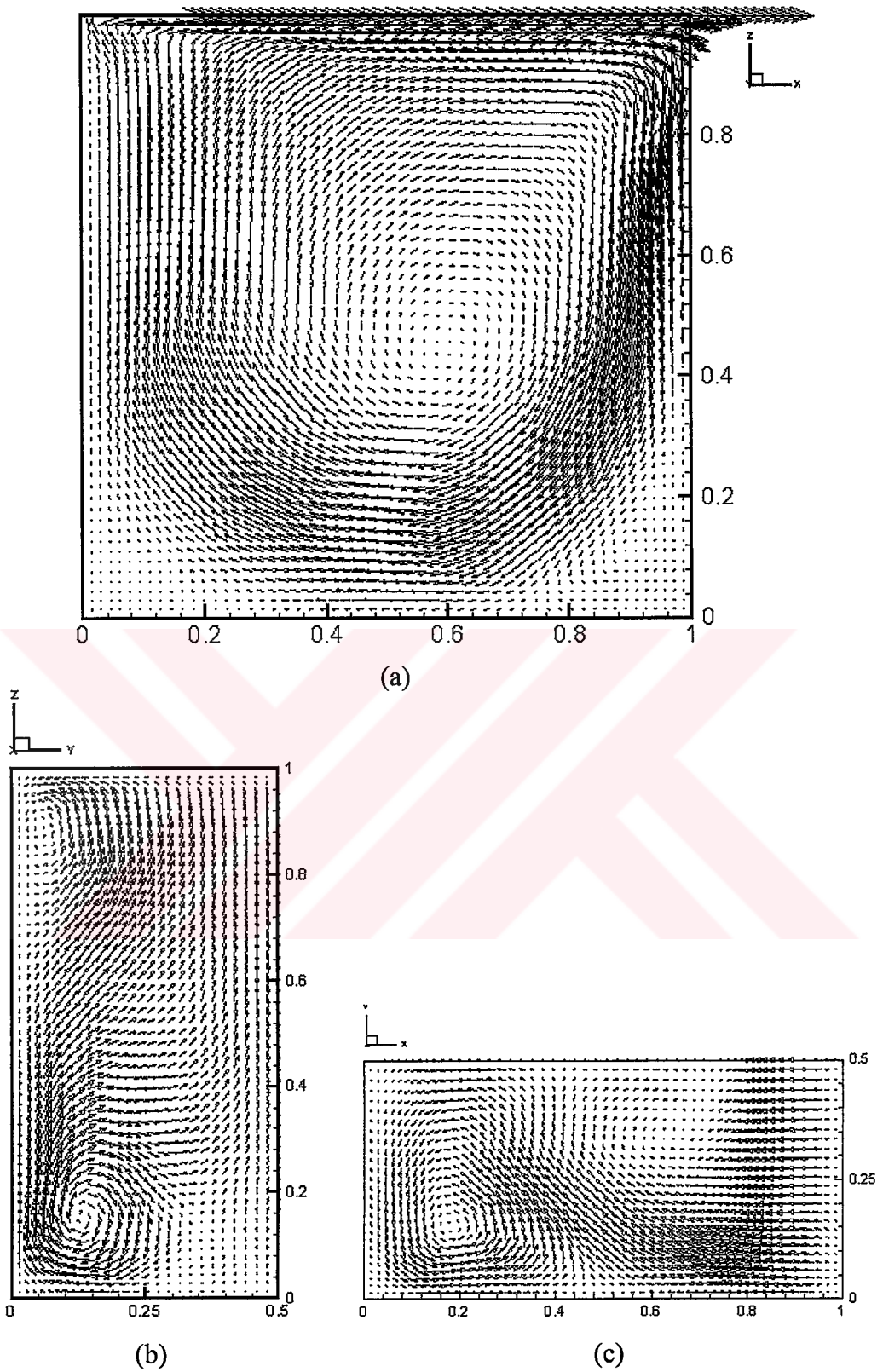


Figure 7.21. 6-domain partitioning 6x(10x28x55) result, third parallelization, velocity vector field: (a) y= 0.5 (symmetry-plane) (b) x = 0.5 (c) z = 0.5 plane, Re=1000.

Comparison of velocity distribution results along the centerlines of the symmetry plane of the cavity, Figure 7.22, shows that, the best agreement with benchmark solution up to now is obtained with the new 6-domain partitioning non-uniform grid with total number of 6 x (10x28x55) nodes.

Velocity distribution, along the horizontal and vertical centerlines of the cubic cavity, obtained with first parallelization, on 2-domain partitioning 2 x (11x11x21) nodes, and with the third parallelization, on 6-domain partitioning are all compared with the benchmark solution of *Ku et al.* in Figure 7.23. As it is seen, the best agreement can be obtained as stretching near the sidewalls of the cavity is increased.

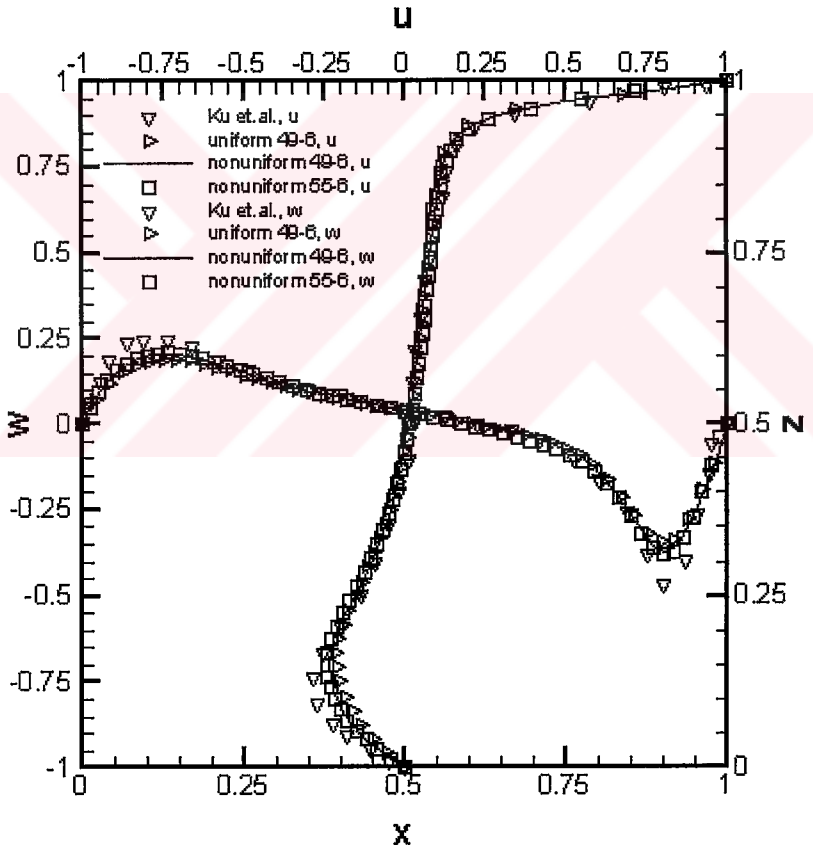


Figure 7.22. Comparison of uniform & non-uniform 6x(9x25x49) and 6x(10x28x55) grids results, velocity distributions along the centerlines of the symmetry-plane, third parallelization, Re=1000.

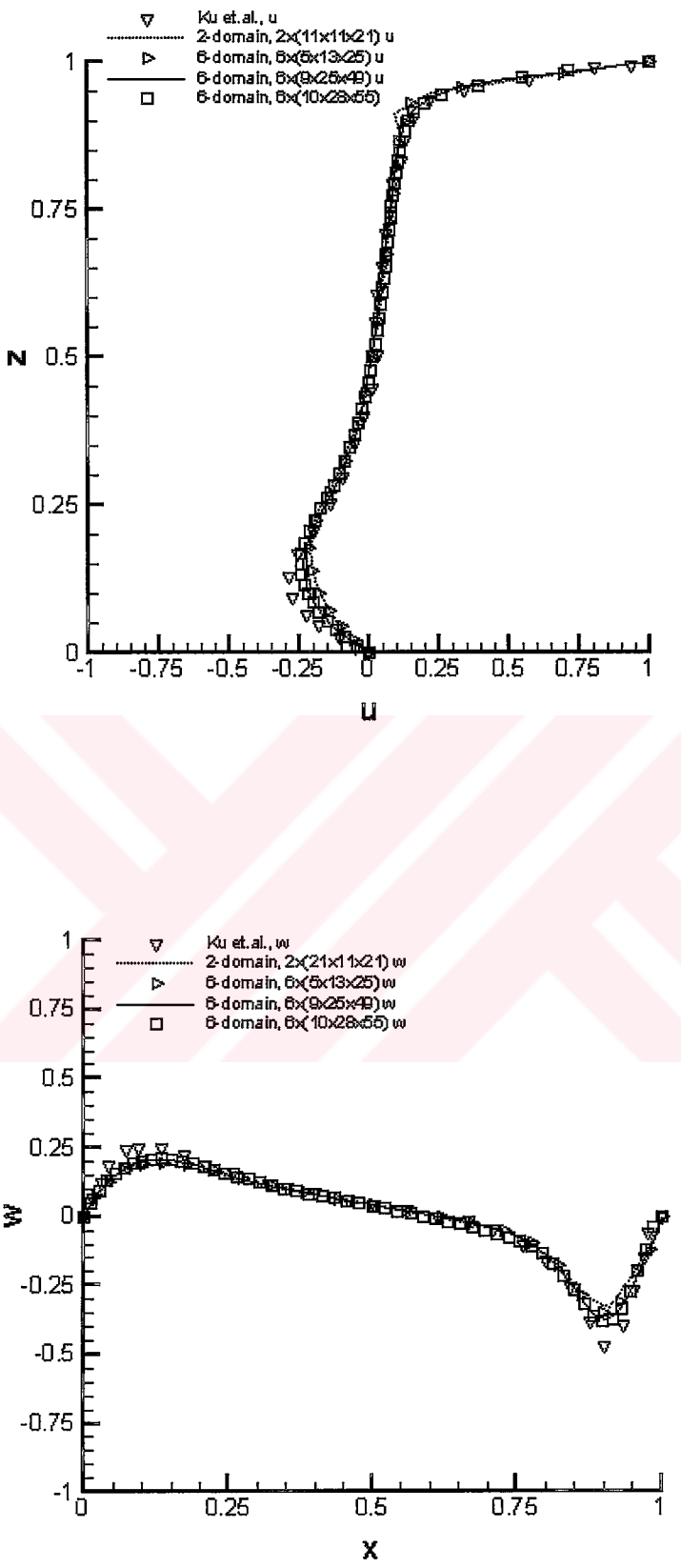


Figure 7.23. Comparison of different grid results for velocity distributions along the centerlines of the symmetry-plane of the cubic cavity, third parallelization, Re=1000.

7.6. Three-dimensional Lid-driven Cavity Flow at Re= 400

The parallel implicit code with the third parallelization algorithm is also calibrated with the lid-driven cavity flow at Reynolds number of 400. 4-domain partitioning computation is done using total number of 4 x (7x13x25) nodes. The solution is converged after 250 time steps with step size of 0.1. The elapsed time of the computation obtained on DEC Alpha XL266 workstations, for Reynolds number of 400, is 2400 seconds. The convergence criteria for half-time velocity and auxiliary potential interface iterations are same with the ones for Reynolds number of 1000.

The pressure contours on the symmetry plane are shown in Figure 7.24. The velocity fields on the symmetry plane and on different cross-sections of the cubic cavity are presented in Figure 7.25 and in Figure 7.26. For Reynolds values of the order of 400, a large primary eddy occupies the core of the flow and two secondary eddies are lodged in the downstream and upstream corners on the bottom surface.

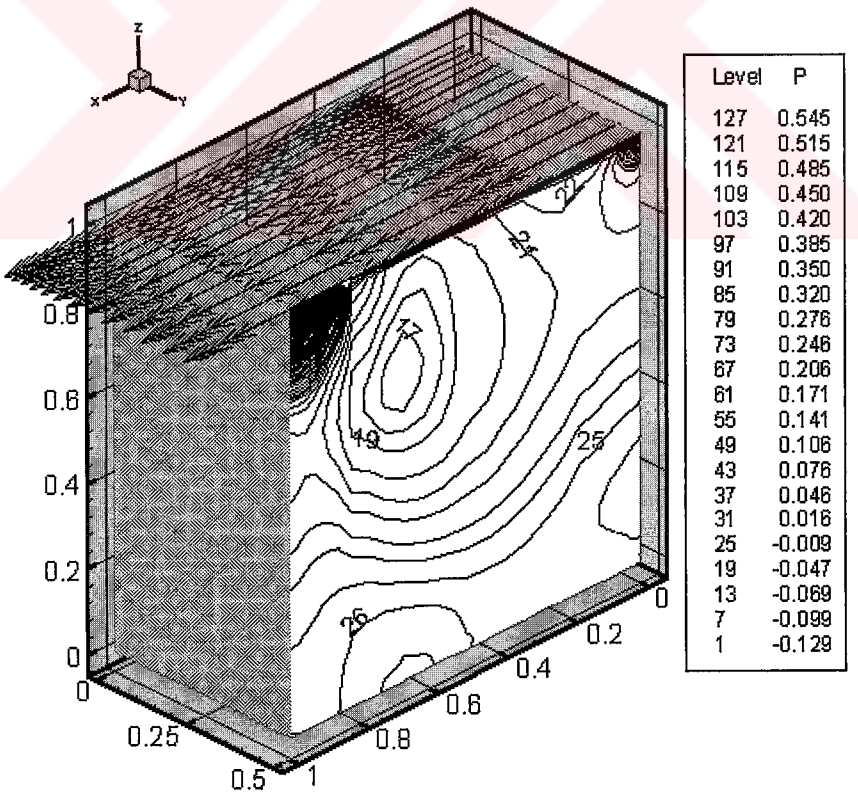
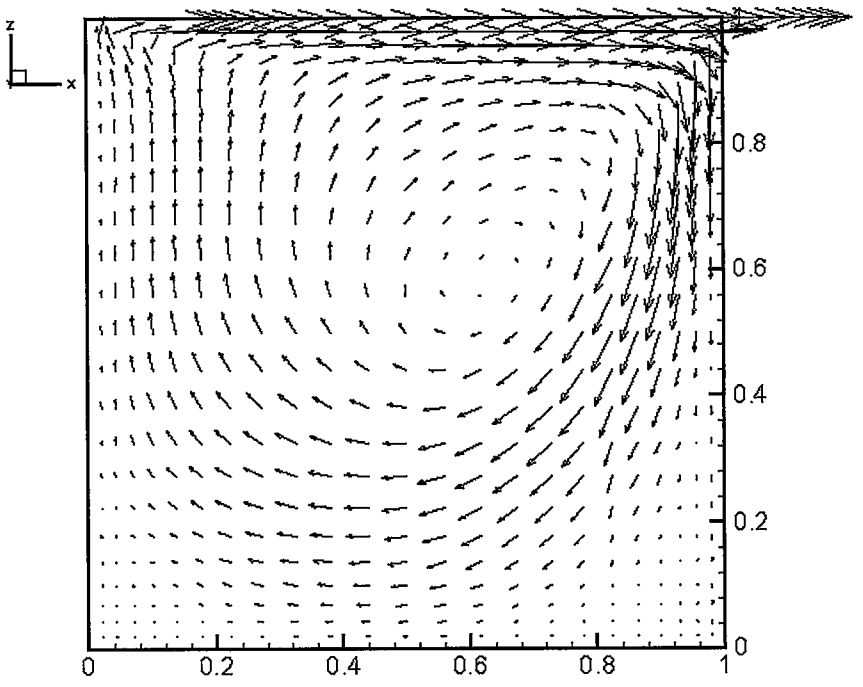
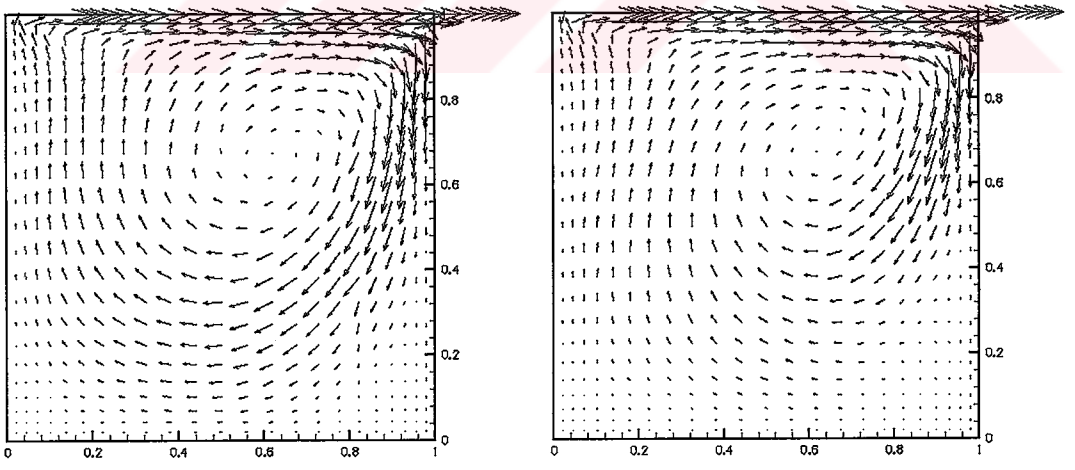


Figure 7.24. Re=400 symmetry-plane pressure contours for the lid-driven cubic cavity flow, third parallelization result, 4-domain partitioning 4x(7x13x25), Δt=0.1, on DEC Alpha XL266 workstations



(a)



(b)

(c)

Figure 7.25. Re=400 velocity vector field, third parallelization results obtained with 4-domain partitioning 4 x (7x13x25): (a) y= 0.5 (symmetry-plane) (b) y = 0.25 (c) y = 0.1 plane, Δt=0.1.

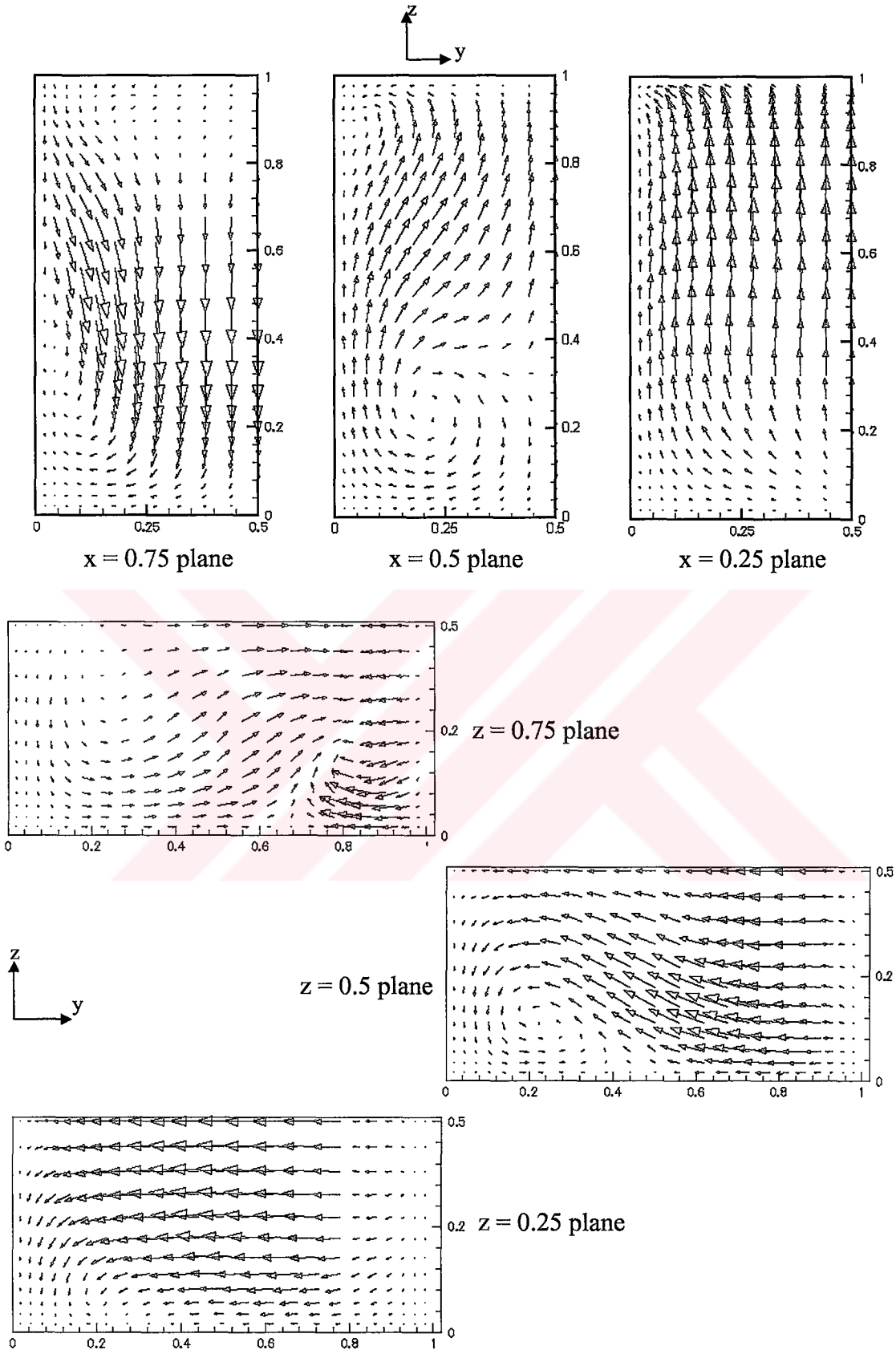


Figure 7.26. Re=400 velocity vector fields on different constant x and constant z planes of the cavity, third parallelization, 4-domain result, Δt=0.1.

Horizontal and vertical velocity vectors along the centerlines of the lid-driven cubic cavity flow with Reynolds number of 400 are compared with two benchmark solutions, Ku *et al.*, [71], and *Babu and Korpela*, [72], in Figure 7.27. The reference Ku *et al.*, [71], results are obtained with pseudo-spectral method and *Babu and Korpela*, [72], results are obtained with finite-difference calculations carried out on a 63 x 33 x 63 uniform grid, with a sequential computations on a Cray.

The velocities from our parallel implicit numerical code agree reasonably well with those of Ku *et al.*, and give better agreement than those of *Babu and Korpela*, although the computations in this study for Reynolds number 400 is carried on much more coarse grid, with 4-domain partitioning including total number of 25x 13 x 25 nodes.

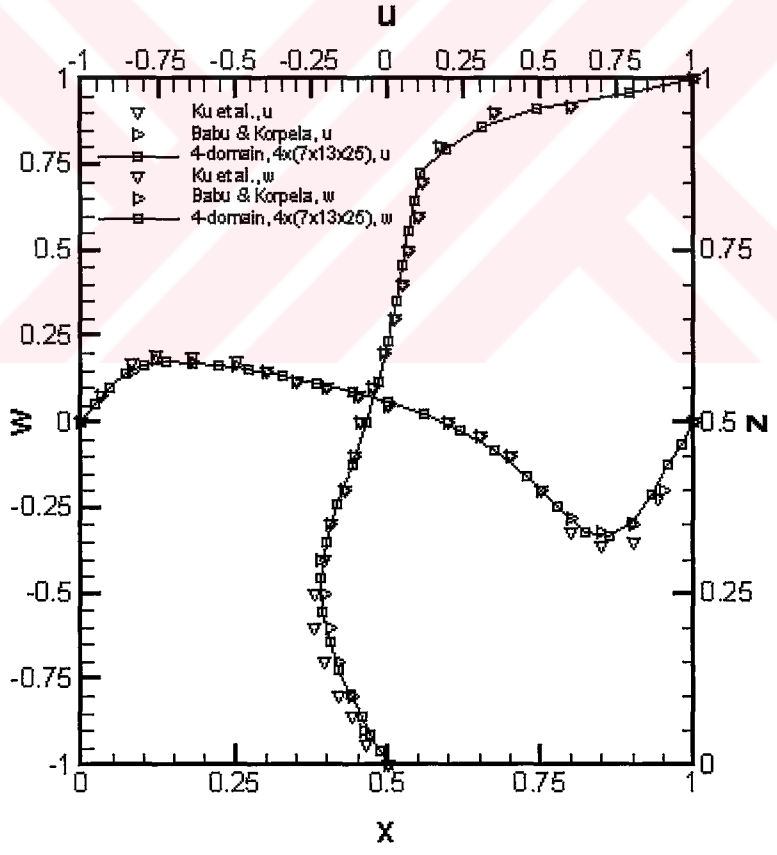


Figure 7.27. Re= 400 result, comparison of velocity distributions along the centerlines of the symmetry-plane of the cubic cavity, third parallelization, 4-domain partitioning 4 x (7x13x25), Δt=0.1.

8. COMPUTATIONS ON SGI O3000 SYSTEM & SPEED-UP ANALYSIS

8.1. Introduction

All parallel studies were continued on a new parallel system after this point. The computations on DEC-Alpha workstations were stopped, because the 2 DEC-Alpha 166 WS computers, which can be used as a master processor for post and pre-processing, broke down. All the parallel data, results and the parallel code were moved to a new system, SGI Origin 3000 system (see Figure 5.1b). Unfortunately, about two months time has been lost to get the data left on the disturbed Alpha stations, to transfer them to the new system, to become familiar to this system and to make the necessary modifications to be able to run the parallel code. The new system, PVM software and new directories were introduced to the parallel code written in this study.

Finally, SGI Origin 3000 is utilized with 8 processor running Unix operating system. The PVM is employed for parallel processing. The sequential code, the parallel code with 2, 4 and 6-domain partitioning results are obtained on the new system successfully and compared in terms of computation time with the solutions on DEC-AlphaXL266 cluster of workstations. These comparisons for lid-driven cubic cavity flow with Reynolds number 1000 and with 6-domain partitioning can be seen in Table 8.1 and Table 8.2.

8.1.1 Comparisons of parallel efficiencies of the first and third parallelization

The parallel results obtained on this new system prove the conclusions made in section 7.4. The master processor speed is important and if all the processors are same in quality and speed the work to solve a problem should be distributed equally on each processor. If the communication speed is high between the master and slave

processors, the first parallel implementation in this study, where the data transference exists only between the master and the slave processors and no communication between the neighboring slaves, is expected to be more efficient in parallelization, as will be proved in the following results. The computational time results obtained on SGI O3000 system are much shorter than the one obtained with DEC-AlphaXL266 workstations; the optimizer of the new system is much better than the one used to get the executable file for the computations done in the previous chapters.

Table 8.1. Comparison of CPU time results carried on SGI O3000 & DEC-Alpha WS, first parallelization method carried on 6-domain partitioning.

1st parallel implementation	LID-DRIVEN CUBIC CAVITY (Re=1000, Δt=0.1) CPU TIME (s)	
Processors	SGI ORIGIN 3000	DEC-AlphaXL266 WS
Master	62	3423
1st slave	108	813
2nd slave	114	1051
3rd slave	114	1063
4th slave	114	648 (on vm06 processor)
5th slave	114	1075
6th slave	108	805

The sequential implicit code is also run on SGI ORIGIN3000 for lid-driven cavity with Reynolds number 1000 carried on 21x11x21 grid points, and the elapsed time is found as 639 seconds, although the elapsed time is 3738 seconds on DEC-AlphaXL266 cluster of workstations. SGI ORIGIN3000 system is approximately 6 times faster than the cluster of workstations. The elapsed time of sequential computation on 25x13x25 grid points is 1934 seconds on SGI O3000 system.

As a result of parallel studies on SGI ORIGIN 3000 system, the first parallel implementation presented in Chapter 5 (flow chart in Figure 5.3) is found out to be the fastest one among the second and third parallel implementations explained in Chapter 7. The second and third parallel implementations are doing the load balance

in order to decrease the work of the master processor and to decrease the efficiency loss because of the slow master processor DEC-Alpha 166. This method increased the speed-up obtained with DEC-Alpha cluster of workstations.

All the three methods are investigated again on SGI Origin 3000 and the first parallel implementation, where the master processor is responsible for the pre- and post-processing of the interface data in the domain decomposition method, is found out to be the most efficient. All 8 processors are exactly equal, the load balancing is not needed, the master and communication speed is high and the master does the steepest descent calculations efficiently and so decreases the amount of work done by the slaves and the slaves do not wait for the master processor. Finally, the overall computational time decreases and speed-up increases with the first parallelization method, compared to that of third parallelization, on this new system.

The computational time for 2, 4 and 6-domain partitioning solutions and comparisons of first and third parallel implementations on SGI ORIGIN 3000 are continued in Table 8.2 – Table 8.6, for lid-driven cavity flow with Reynolds number 1000 and with total number of 25x13x25 nodes.

Table 8.2. Comparison of elapsed time results carried on SGI O3000 & DEC-Alpha WS, obtained with the first and third parallelization computations.

Total number of nodes 25x13x25	LID-DRIVEN CUBIC CAVITY (Re=1000, Δt=0.1) ELAPSED TIME (s)		
Number of domains	2	4	6
1st parallel implementation SGI ORIGIN 3000	546	237	227
1st parallel implementation DEC-AlphaXL266	4448	3229	4979
3rd parallel implementation SGI ORIGIN 3000	561	262	258
3rd parallel implementation DEC-AlphaXL266	4105	1719	1579

Table 8.3. Comparison of master CPU time results, first and third parallelization, on SGI ORIGIN 3000.

SGI ORIGIN 3000	LID-DRIVEN CUBIC CAVITY (Re=1000, Δt=0.1) MASTER CPU TIME (s)		
Number of domains	2	4	6
1st parallel implementation	7.929	31.780	62.270
3rd parallel implementation	0.018	0.021	0.024

Table 8.4. Comparison of slave CPU time results obtained with first and third parallelization on SGI ORIGIN 3000, with 2-domain partitioning

SGI ORIGIN 3000	CUBIC CAVITY, 2-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)	
Slave order	1st	2nd
1st parallel implementation	531.09	530.07
3rd parallel implementation	538.84	536.70

Table 8.5. Comparison of slave CPU time results obtained with first and third parallelization, on SGI ORIGIN 3000, with 4-domain partitioning

SGI ORIGIN 3000	CUBIC CAVITY, 4-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)			
Slave order	1st	2nd	3rd	4th
1st parallel implementation	180.01	185.47	185.68	180.08
3rd parallel implementation	191.49	195.79	196.02	205.09

Table 8.6. Comparison of slave CPU time results obtained with first and third parallelization on SGI ORIGIN 3000 with 6-domain partitioning

SGI ORIGIN 3000	CUBIC CAVITY, 6-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)					
Slave order	1st	2nd	3rd	4th	5th	6th
1st parallel implementation	108.18	114.13	114.07	114.75	114.25	108.65
3rd parallel implementation	120.89	127.20	126.40	126.27	126.24	155.80

Speed-up (elapsed time) and speed-up (slave CPU time) values for the parallel results, obtained on SGI ORIGIN 3000, are calculated with respect to the 2-domain solution and presented in Table 8.7. When the processors are equal, parallel processing necessitates the equal amount of work shared between each processor including the master one.

If the CPU time is considered, the speed-up reaches up to 2.95 with 4-domain computation and up to 4.91 with 6-domain computation. But the speed-up values based on elapsed time is smaller, because of the lost time during the communications between the processors; as the number of domains increases, interface values to be transmitted and also the number of interface iterations increase, too. As explained in Chapter 7, with 8-domain partitioning, speed-up could be more efficient than the 6-domain case, because half-bandwidth decreases more and so the data to be processed becomes less.

The third parallelization is not efficient on SGI ORIGIN 3000 system, because it makes the load balancing suitable for the DEC-AlphaXL266 system, it eliminates the master processor and loads its work to the fastest one and communications are mostly continued between the neighboring slaves and the fastest slave. But this kind of loading increases the work of last slave on SGI ORIGIN 3000, which does not differ in speed from the other slave processor. So the computation on this processor takes longer. Related to this, the computations on the other processors delay. By the first parallelization method, the master processor shares the domain decomposition calculations with the slave processor, so the speed-up increases if the sub-domains increase.

Table 8.8 compares the speed-up (elapsed time) values reached at SGI ORIGIN 3000 and those at DEC-AlphaXL266 cluster of workstations. As explained in Chapter 7, the speed-up increases successfully on DEC-AlphaXL266, if the slow master processor DEC-Alpha166 shares no work during the parallel calculations and communications and if the last slave vm06 DEC-AlphaXL266 processor, which is faster than the other processors used as a slave in the parallel computations, does the task of calculating and sending the domain decomposition coefficients at each interface iteration cycle.

Table 8.7. Comparison of speed-up results obtained with first and third parallelization on SGI ORIGIN 3000 for lid-driven cubic cavity flow

SGI ORIGIN 3000	CUBIC CAVITY (Re=1000, Δt=0.1)		
Number of domains	2	4	6
Speed-up (slave CPU time) 1st parallel implementation	1	2.95	4.91
Speed-up (elapsed time) 1st parallel implementation	1	2.30	2.40
Speed-up (elapsed time) 3rd parallel implementation	1	2.14	2.17
Speed-up (elapsed time) Explicit Method, [51]	1	2.17	

The reason of why the speed-up results obtained on SGI O3000 are smaller than those on DEC-Alpha workstations can be explained as follows; if the processor, which collects all the interface data and process them (as explained in domain decomposition method presented in Chapter 5), is faster than the slave processors, then the elapsed time decreases more and overall speed-up increases.

If the results are investigated well, it can be seen that, the speed-up is increased when vm06 machine, which is faster than the other slaves, is loaded with the work of pre- and post-processing. On SGI O3000 system, the speed of master processor does not differ from the slave processors, so the speed-up is not as much as in DEC-Alpha workstations. When the number of partitioning increases, the interface data increases; then, especially the speed of master processor becomes very important factor on speed-up.

All of the results up to now, shown in the tables of this chapter, are obtained on total number of 25x13x25 grid points. The lid-driven cubic cavity result with total number of 6 x (10x28x55) grid points with 6-domain partitioning is obtained successfully at SGI ORIGIN3000 system. The elapsed time is 9774 seconds and slave CPU time is 8855, 8955, 8875, 8959, 8874 and 8941 seconds and 190 seconds for the master processor. Reynolds number is 1000 and the step size is Δt=0.05.

Relative CPU time of computations with first parallelization method, on SGI O3000 system, with 2, 4 and 6-domain partitioning (total number of 25x13x25 nodes) is graphed in Figure 8.1.

Table 8.8. Comparison of speed-up results obtained on SGI ORIGIN 3000 & on DEC-AlphaXL266 WS for lid-driven cubic cavity flow

	CUBIC CAVITY (Re=1000, Δt=0.1)		
Number of domains	2	4	6
Speed-up (elapsed time) 1st parallel implementation SGI ORIGIN 3000	1	2.30	2.40
Speed-up (elapsed time) 1st parallel implementation DEC-AlphaXL266	1	1.37	0.89
Speed-up (elapsed time) 3rd parallel implementation DEC-AlphaXL266	1	2.39	2.60

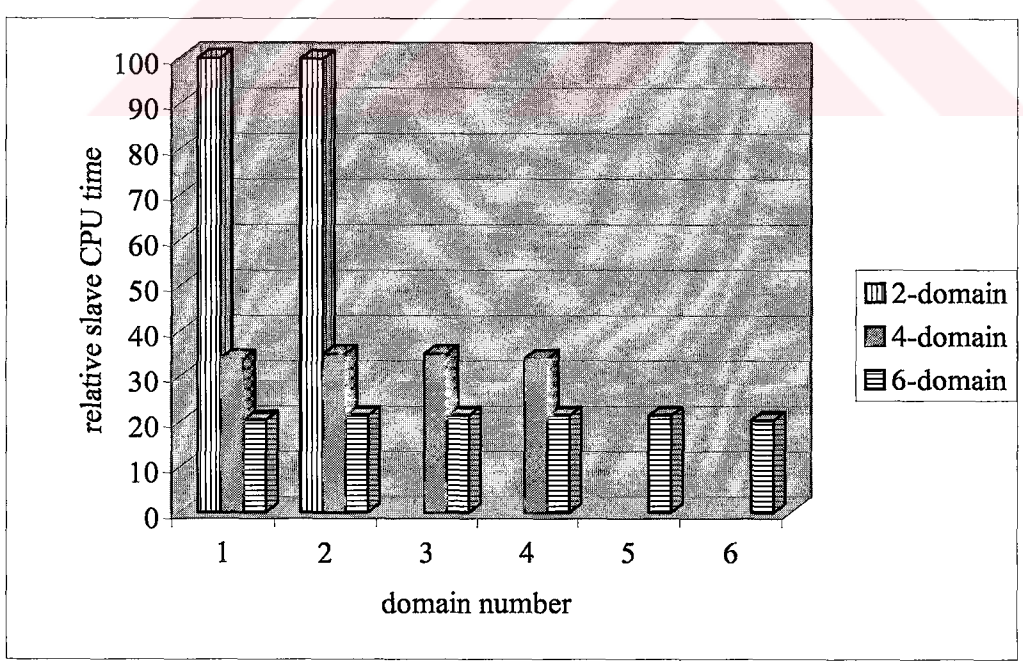


Figure 8.1. Relative CPU time of each slave processor computation carried on SGI ORIGIN 3000, lid-driven cubic cavity flow, 2, 4 and 6-domain partitioning, first parallelization, Re=1000, Δt=0.1.

8.2 Full Geometry Solution: lid-driven cubic cavity flow

Lid-driven cubic cavity problem with Reynolds number 1000 is solved for full geometry with total number of 25x25x25 grid points. Flow is symmetric in this test case, all the solutions up to consider only the symmetric part and in this section, it is aimed to show that the flow is symmetric and the symmetry solution is wanted to be check by the comparisons with the full geometry solution on the symmetry plane. Symmetry boundary condition applied on y= 0.5 plane is removed from the code, and no-slip boundary conditions are applied on the walls, including y= 1 plane wall.

The lid velocity is 1.0 and length of all the walls is 1. The domain is divided into 6 sub-domains and parallel computation with the first parallel implementation carried on SGI ORIGIN3000 is completed successfully. The bandwidth of the coefficient matrices is 132. Time step is taken as 0.1 and the solution is advanced up to the dimensionless time level of 25. The grid and the pressure field at the center plane (symmetry plane), y =0.5, are shown in Figure 8.2 and in Figure 8.3, respectively.

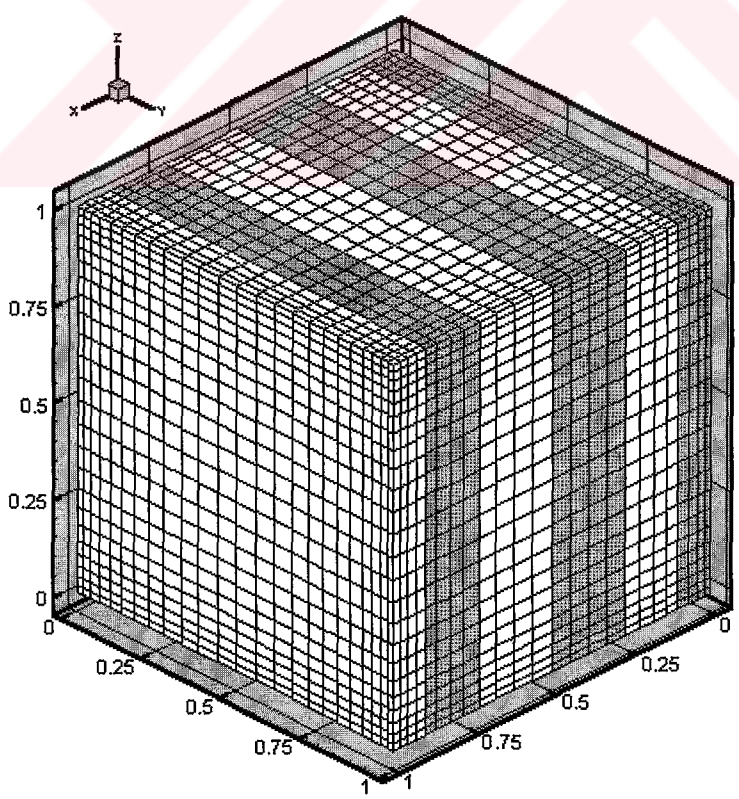


Figure 8.2. Full geometry: lid-driven cubic cavity, 6x(7x25x25) grid points

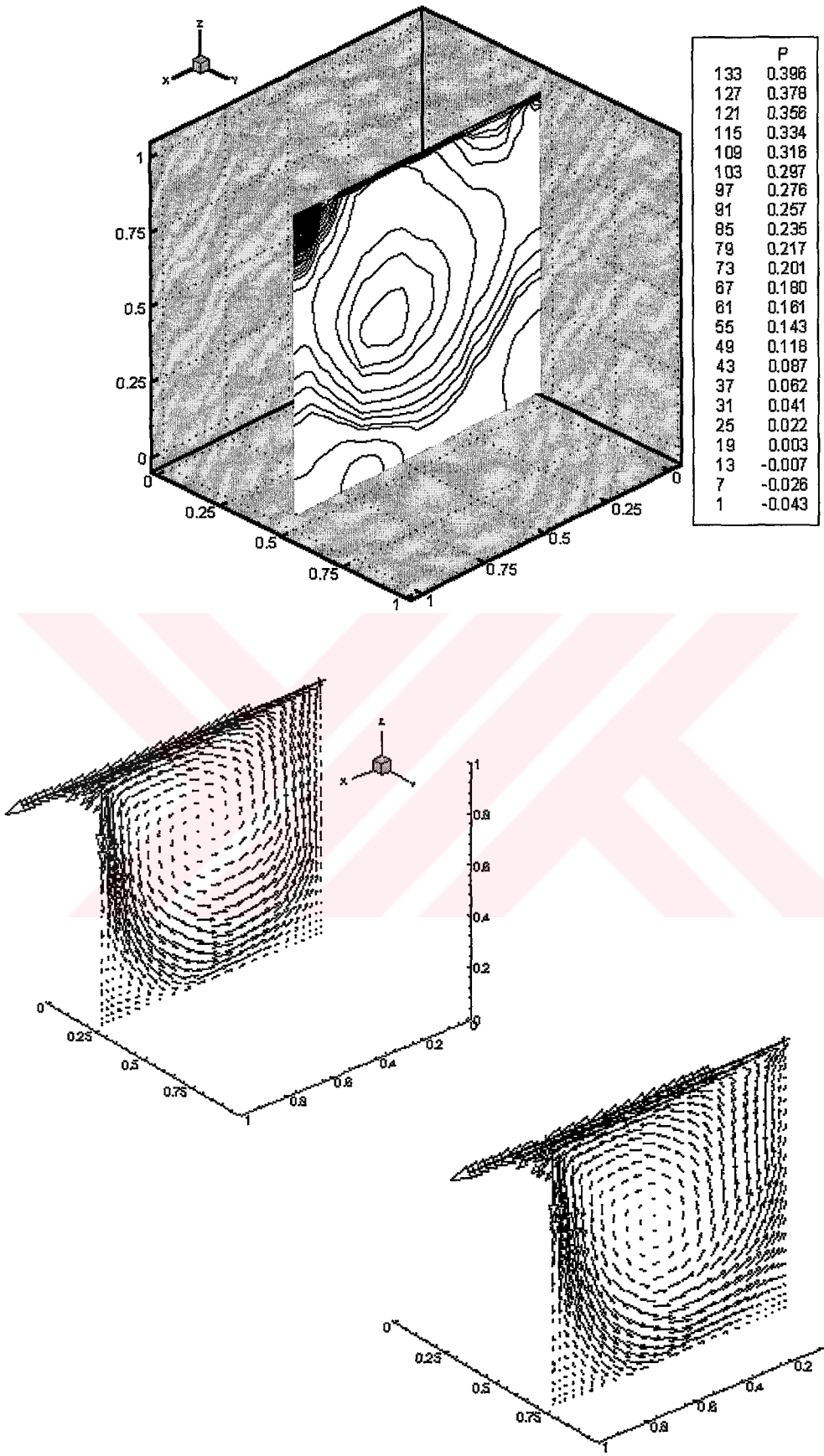


Figure 8.3. Full geometry results, lid-driven cavity flow: (a) pressure contours on y=0.5 plane and velocity field on (b) y=0.25 (c) y=0.75 plane, Re = 1000.

Velocity and pressure values at this plane are checked and found exactly the same as the results found from the computations including only the symmetry part of this problem. The velocity vector field on different constant y planes of the cubic cavity is shown in Figure 8.3. Symmetry of the three dimensional lid-driven cubic cavity is shown in Figure 8.4. Two stream-traces of the flow is drawn, starting coordinates of them are very close to the walls as written under the figure.

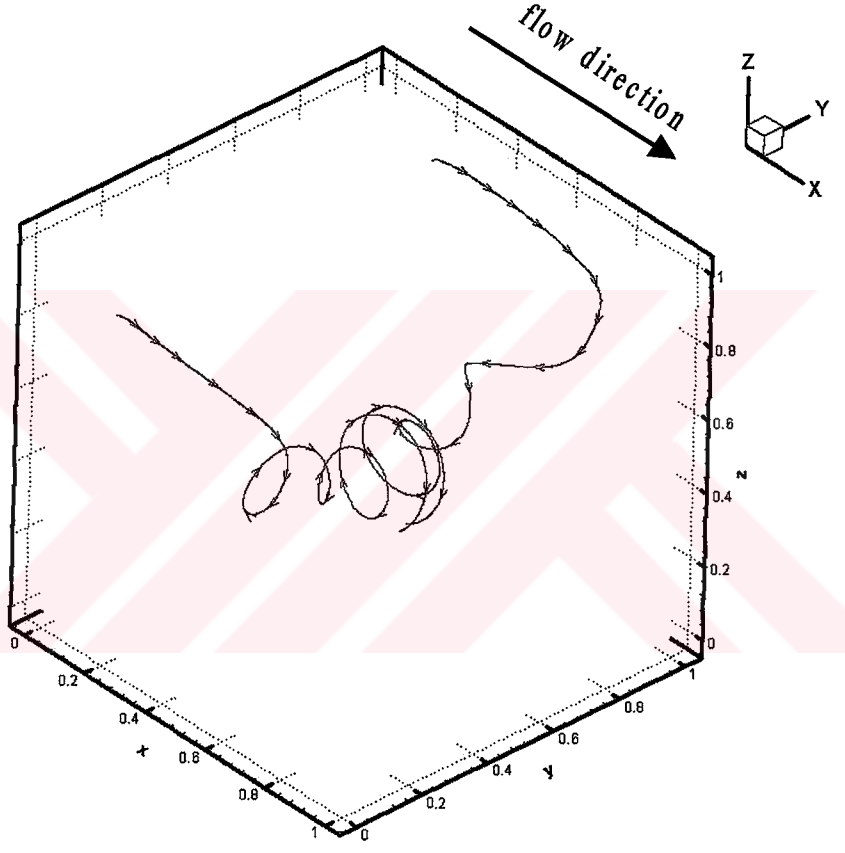


Figure 8.4. Two stream-traces of the three dimensional lid-driven cavity flow: initial points coordinates: (x, y, z) = (0.2, 0.02, 0,9) and (0.2, 0.98, 0.9)

9. MODIFICATIONS IN DOMAIN DECOMPOSITION EQUATIONS & IMPROVEMENT IN SPEED-UP

9.1. Introduction

Speed-up analysis up to this chapter shows that, the speed-up, based on the elapsed time, is 2.30 with 4-domain, and 2.40 with 6-domain partitioning computations, which are ideal values, but can be improved. As indicated in Chapter 7, although the interface iterations for the half-time step velocity field mostly count 8 iterations, the interface auxiliary potential iterations reach up to 100. There is no need to decrease the velocity iterations, even decreasing them to less than 8 can cause accuracy problems, but the auxiliary potential one counts to many, causing the problems in speed-up. As the interface number increases, the interface data to be processed for the parallel solution and the number of iterations increase and all the master and the slave processors spent longer time for the interface calculations, also the time lost for the communications increases.

It is believed that, if the number of interface auxiliary potential iterations can be reduced, than the computational time can be decreased and speed-up improvement can be achieved. For this purpose, the researches are started, from the beginning of first parallel results, to find a way to reduce the auxiliary potential interface iteration. One of the trials is to calculate the pressure with a direct solution at each sub-domain and to find the pressure interface values by domain decomposition method, instead of calculating the auxiliary potential function, which satisfies the Poisson's equation and defined in equation (3.18) in terms of the difference of the pressure at old and new time levels. But this method could not achieve reducing the interface iterations.

With the latest modifications in domain decomposition equations, as will be explained in section 9.3, the most important goal is achieved, the interface auxiliary potential iterations are reduced sufficiently and super-linear speed up is obtained.

9.2. Direct Pressure Formulation

As explained in Chapter 3 and Chapter 4, the parallel CFD code written in this thesis uses the direct auxiliary potential function formulation. The pressure at each time step is obtained via an auxiliary potential which satisfies the Poisson's equation and the domain decomposition technique is applied for the efficient parallel solution of the momentum, equation (3.17) and Poisson's equation for the auxiliary potential function, equation (3.20).

So, it is decided to search whether the interface iterations can be reduced more using the direct pressure formulation instead of using the auxiliary potential function formulation. The code is modified, the solution steps are indicated below:

i. Equation (3.17) is solved to find the half-time step velocity field at new time level n+1/2 with domain decomposition method (Chapter 5),

ii. Discretized form of equation (3.6) together with equation (3.7) in Chapter 3 gives;

$$\frac{1}{2}\mathbf{A}(p^{n+1}-p^{n})\Delta t = -2\mathbf{E}_{\alpha}\mathbf{u}_{\alpha}^{n+1/2}, \tag{9.1}$$

If the above equation is rearranged, Poisson's equation for the pressure is obtained as follows:

$$\mathbf{A}\,p^{n+1} = \mathbf{A}\,p^{n} - \frac{1}{\Delta t}4\mathbf{E}_{\alpha}\mathbf{u}_{\alpha}^{n+1/2}. \tag{9.2}$$

Equation (9.2) is solved implicitly. The pressure at the new time level (n+1) is solved using the old time level (n) pressure value and the half time step (n+1/2) velocity field. Pressure is calculated directly by the subroutine SYM, that is oriented to systems with symmetric and banded coefficient matrix. Domain decomposition technique is employed for the efficient parallel solution of the Poisson's equation for the pressure equation, equation (9.2). At the initialization part of the domain decomposition method (see Chapter 5), the following equations are solved for the pressure,

Initialization: Equation (9.2) is solved in each domain Ω_i with boundary of $\partial\Omega_i$ and interface S_j, with vanishing Neumann boundary condition on the domain interfaces.

$$\overline{\mathbf{A}}\mathrm{y_i} = \mathrm{f_i} \qquad \text{in } \Omega_i \qquad\qquad \mathbf{A}\,p_i^{o} = \mathrm{f_i^{o}} = 0.$$

$$\mathbf{A}\,p_i^{k+1} = \mathrm{f_i^{k+1}} = \mathbf{A}\,p_i^{k} - \frac{1}{\Delta t}4\mathbf{E}_\alpha \mathbf{u}^{k+1/2} \qquad \mu^{o} = 0 \text{ and } \mathrm{g^{o}} = (\mathrm{y_1} - \mathrm{y_2})S_j$$

$$\mathrm{y_i} = \mathrm{g_i} \qquad \text{on } \partial\Omega_i \qquad\qquad w^{o} = \mathrm{g^{o}}$$

$$\frac{\partial \mathrm{y_i}}{\partial \mathrm{n_i}} = 0 \qquad \text{on } S_j$$

where, $\overline{\mathbf{A}} = \mathbf{A}$ is the stiffness matrix in equation (3.20) and $\mathrm{y_i} = p_i$ stands for the nodal pressure, $\mathrm{f_i}$ is the right hand side of the Poisson's Equation, equation (9.2), and subscripts i and k denote the sub-domain number and the time level, respectively.

After the initialization part is completed, unit problem and finalization parts are completed as explained in Chapter 5, and so the nodal pressure is calculated in the domains with the correct Neumann boundary conditions.

iii. The element pressure is computed using the nodal pressure as,

$$p_e = \frac{1}{\mathrm{vol}(\Omega_e)} \int_{\Omega_e} \mathrm{N_i}\, p_i \, d\Omega_e, \qquad i = 1,\ldots\ldots,8.$$

Via the pressure averaging, nodal pressure values are obtained at each sub-domain.

iv. The discretized form of equation (3.9) gives the full time-step level velocity field for the direct pressure formulation;

$$\mathbf{M}\mathbf{u}_\alpha^{n+1} = 2\mathbf{M}\mathbf{u}_\alpha^{n+1/2} - \mathbf{M}\mathbf{u}_\alpha^{n} - \frac{\Delta t}{2}\mathbf{E}_\alpha (p^{n+1} - p^{n}). \tag{9.3}$$

Initially, the direct pressure formulation is applied to the sequential implicit code for lid-driven cubic cavity flow with Reynolds number 1000. Total number of grid points is 21x11x21 and time step is 0.1. No significant difference is found out between the results computed with the direct auxiliary potential formulation and the direct pressure formulation. The results are shown in Table 9.1.

Table 9.1. Comparison of auxiliary potential function & direct pressure formulations with lid-driven cubic cavity flow, sequential computation on SGI O3000.

21x21x11 grid points SEQUENTIAL RUN	LID-DRIVEN CUBIC CAVITY (Re=1000, Δt=0.1)	
	AUXILIARY POTENTIAL FORM.	PRESSURE FORM.
ELAPSED TIME (s)	639.0	641.0
CPU TIME (s)	638.8	639.6

The parallel 2 and 4-domain partitioning solutions show that; domain decomposition technique employed for the parallel solution of the Poisson's Equation with auxiliary potential function formulation takes less computational time than that with the direct pressure formulation. No other difference is observed except the computational time. Results are compared in Table 9.2. At the computations, the convergence check criteria for the velocity and pressure iterations at the interfaces is taken as $\varepsilon_{\mathbf{u}}=10^{-5}$ and $\varepsilon_{\phi}=10^{-6}$, respectively, for the direct auxiliary potential formulation. For the direct pressure formulation $\varepsilon_{\mathbf{u}}=10^{-5}$ and $\varepsilon_{p}=10^{-4}$.

Table 9.2. Comparison of auxiliary potential function & direct pressure formulations with lid-driven cubic cavity flow, first parallelization method, SGI O3000.

Total 25x13x25 grid points PARALLEL RUN	LID-DRIVEN CUBIC CAVITY (Re=1000, Δt=0.1) ELAPSED TIME (s)	
Partitioning	AUXILIARY POTENTIAL FORM.	PRESSURE FORM.
2-DOMAIN SOLUTION	546	577
4-DOMAIN SOLUTION	237	261

Interface iterations could not be reduced with the direct pressure formulation but with the following studies, the goal is reached.

9.3. Modification of Domain Decomposition Equations and Improvement in Speed-up

In this section, the direct auxiliary potential formulation is applied to solve Poisson's Equation, equation (3.20). But modifications are done in the domain decomposition equations, explained in Chapter 5, to reduce interface auxiliary potential iterations. The main idea is to set the interface Neumann boundary conditions at the beginning of each time level, equal to the converged value which is found out at the end of the interface iterations, but not zero as it is done in the domain decomposition equations described in Chapter 5.

With this idea, it is expected that, interface computations can be converged more easily, the number of iterations can be reduced and speed-up can be improved. Related to this goal, equations are modified and the new form of the domain decomposition steps are rewritten as follows:

Initialization: Poisson's equation (3.20) is solved with a direct solution method, separately in each domain Ω_i with boundary of $\partial\Omega_i$ and interface S_j, with vanishing Neumann boundary condition on the domain interfaces,

Only, at the first time level: $\mu^k = 0$ and $aw^k = 0$,

$$\overline{\mathbf{A}}\mathrm{y}_i = \mathrm{f}_i \qquad \text{in } \Omega_i, \qquad \mu^o = \mu^k$$

$$\mathrm{y}_i = \mathrm{g}_i \qquad \text{on } \partial\Omega_i, \qquad \mathrm{g}^o = aw^k - (\mathrm{y}_2^o - \mathrm{y}_1^o)S_j$$

$$\frac{\partial \mathrm{y}_i}{\partial \mathrm{n}_i} = (-1)^{i-1}\mu^k \qquad \text{on } S_j, \qquad w^o = \mathrm{g}^o$$

where subscript k denotes the interface iteration level at which convergence is obtained, and subscript o stands for the initial interface iteration level at the each time-step level of the computation.

Unit Problem: Inside the interface iteration cycle, each slave starts to solve the unit problem using the direct solution, together with the initialized Neumann condition received from the master at the end of the initialization step.

$$\overline{\mathbf{A}}\mathrm{x}_{\mathrm{i}}^{n} = 0 \qquad \text{in } \Omega_{\mathrm{i}}$$

$$\mathrm{x}_{\mathrm{i}}^{n} = 0 \qquad \text{on } \partial\Omega_{\mathrm{i}}$$

$$\frac{\partial \mathrm{x}_{\mathrm{i}}^{n}}{\partial \mathrm{n}_{\mathrm{i}}} = (-1)^{\mathrm{i}-1} w^{n} \qquad \text{on } S_j$$

Each slave sends its new computed interface values to the master processor at the end of the unit problem and master starts post-processing of the received data from the slaves. The interface iteration cycle starts up at the master processor. Master optimizes the interface nodal values of the unknowns using the following '*steepest descent*' procedure.

Steepest Descent: Interface iteration cycle goes on until the master processor finds correct Neumann conditions at the interfaces, μ^{k}, at the end of its post processing of the interface data received from the slave processors.

$$aw^{n} = \left(x_1^{n} - x_2^{n}\right) S_j$$

$$\mathrm{g}^{n+1} = \mathrm{g}^{n} - \beta^{n} aw^{n} \qquad w^{n+1} = \mathrm{g}^{n+1} + \mathrm{s}^{n} w^{n}$$

$$\beta^{\mathrm{n}} = \frac{\sum\limits_{j} \int\limits_{S_j} \left|\mathrm{g}^{n}\right|^{2} ds}{\sum\limits_{j} \int\limits_{S_j} (aw^{n}) w^{n} ds} \qquad \mathrm{s}^{n} = \frac{\sum\limits_{j} \int\limits_{S_j} \left|\mathrm{g}^{n+1}\right|^{2} ds}{\sum\limits_{j} \int\limits_{S_j} (\mathrm{g}^{n})^{2} ds}$$

$$\mu^{n+1} = \mu^{n} - \beta^{n} w^{n}$$

where n denotes the interface iteration level.

Convergence check: If $\left|\mu^{n+1}-\mu^{n}\right| < \varepsilon_{\phi}=10^{-6}$ then $aw^{n} = aw^{k}$

$\mu^{n+1} = \mu^{k}$

If the convergence is satisfied, interface iteration cycle stops and the converged μ^{k} and aw^{k} values are kept to be used in the following time level at the initial interface iteration level. At the following time level, each slave processor solves equation (3.20) together with the correct Neumann condition of the previous time level, $\frac{\partial y_i}{\partial n_i} = (-1)^{i-1}\mu^{k}$ on the interface, and the slave processor starts to solve unit problem together with the initial interface Neumann condition of $\frac{\partial x_i^{o}}{\partial n_i} = (-1)^{i-1} w^{o}$, where $w^{o} = aw^{k} - (y_2^{o} - y_1^{o})S_j$.

Finalization: Having obtained the correct Neumann boundary condition for each interface, the Poisson equation (3.20) is solved via the direct solution method at each sub-domain, together with the correct interface conditions,

$$\overline{\mathbf{A}} y_i = f_i \qquad \text{in } \Omega_i,$$

$$y_i = g_i \qquad \text{on } \partial\Omega_i,$$

$$\frac{\partial y_i}{\partial n_i} = (-1)^{i-1}\mu^{k} \qquad \text{on } S_j.$$

9.3.1 Results and Discussion

The modified domain decomposition method is calibrated with the three-dimensional lid-driven cubic cavity flow, with Reynolds number of 1000. Computations are carried on the existing 2, 4 and 6-domain partitioning grids with total number of 25x13x25 nodes and also on the 6-domain partitioning grid with total number of 6 x (10x28x55) nodes. The results obtained on SGI O3000 are compared with the previous results obtained with first parallelization method on SGI O3000 system and with the benchmark solution of Ku *et al.*, [71].

The comparison of horizontal and vertical velocity distributions along the centerlines of the cavity is shown in Figure 9.1 for the 6-domain partitioning case with 6x (10x28x55) nodes. Elapsed time results are compared with the first parallelization runs on SGI O3000 in Table 9.3.

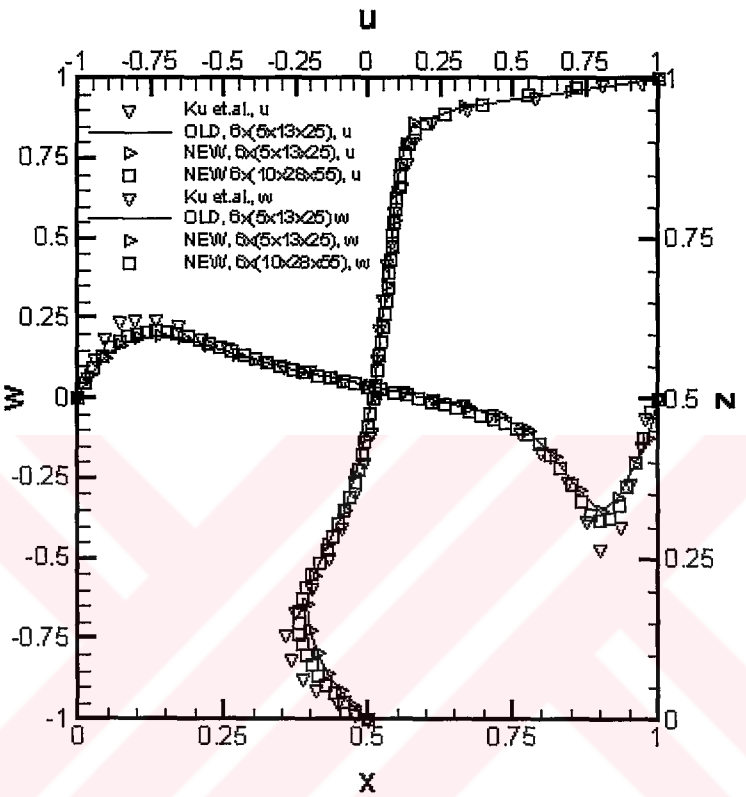


Figure 9.1. New modified domain decomposition result: comparison with the old result obtained by first parallelization method, Re=1000, Δt=0.1.

Table 9.3. Comparison of elapsed time of the results, obtained by the new modified domain decomposition computation, with the first parallelization method, 2, 4 and 6-domain partitionings, on SGI O3000 system

SGI ORIGIN 3000 Total 25x13x25 nodes	CUBIC CAVITY, 6-DOMAIN (Re=1000, Δt=0.1) ELAPSED TIME (s)		
Domain partitioning	2	4	6
First parallelization run	546	237	227
New modification run	471	174	151

Accuracy of the new results does not differ from the first parallelization computations but computational time decreases significantly and speed-up is much higher than that of the previous parallel computations in this study, super-linear speed-up is achieved.

Table 9.4 compares CPU time results obtained with the modified domain decomposition method, carried on total number of 25x13x25 nodes, with the previous results obtained via the first parallelization method on SGI O3000 system.

Table 9.4. Comparison of slave CPU times of the results obtained by the new modified domain decomposition computation with the first parallelization method, 2, 4 and 6-domain partitionings, on SGI O3000 system

SGI ORIGIN 3000 2 x (13x13x25)	CUBIC CAVITY, 2-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)	
Slave order	1^{st}	2^{nd}
First parallelization run	531	530
New modification run	460	460

SGI ORIGIN 3000 4 x (7x13x25)	CUBIC CAVITY, 4-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)			
Slave order	1^{st}	2^{nd}	3^{rd}	4^{th}
First parallelization run	180	185	185	180
New modification run	139	143	143	139

SGI ORIGIN 3000 6 x (5x13x25)	CUBIC CAVITY, 6-DOMAIN (Re=1000, Δt=0.1) SLAVE CPU TIME (s)					
Slave order	1^{st}	2^{nd}	3^{rd}	4^{th}	5^{th}	6^{th}
First parallelization run	108	114	114	114	114	108
New modification run	81	85	85	85	85	81

The interface convergence criteria is same as before, $\varepsilon_{\mathbf{u}}=10^{-5}$ and $\varepsilon_{\phi}=10^{-6}$. The number of interface auxiliary potential iterations decreased approximately % 50.

On the 2-domain partitioning, first parallelization interface inner iteration number for auxiliary potential rises up to 55 iterations, but the modified domain decomposition computation decreases the iteration number to mostly 25. On the 4-domain computation, the iterations count up to 85 with the first parallelization computation and up to 45 with the modified equations. On the 6-domain computation, the interface iteration number counted at the first parallelization and at the modified domain decomposition computations are 105 and 61, respectively. Half-time step velocity interface iterations do not exceed 8.

The slave processors CPU times obtained on 6 x (10x28x55) nodes are also compared with the previous first parallelization results obtained in Chapter 8, the comparison is presented in Table 9.5.

Table 9.5. Comparison of slave CPU times of the results obtained by the new modified domain decomposition computation with the first parallelization method, 6-domain partitioning run, 6 x (10x28x55), on SGI O3000 system

SGI ORIGIN 3000 6 x (10x28x55)	CUBIC CAVITY, 6-DOMAIN (Re=1000, Δt=0.05) SLAVE CPU TIME (s)					
Slave order	1st	2nd	3rd	4th	5th	6th
First parallelization run	8855	8955	8875	8959	8874	8941
New modification run	7221	7310	7222	7312	7218	7305

Since the interface auxiliary potential iterations are reduced sufficiently, the speed up is improved as shown in Table 9.6. With the new algorithm, speed-up (elapsed time) rises to 3.12 and speed-up (slave CPU time) up to 5.69, based on the 2-domain partitioning computation time. Relative CPU times of the computations on each slave processors are graphed for this latest improvement, in Figure 9.2.

The modified domain decomposition equations also work well with the direct pressure formulation, which is explained at the beginning of this chapter; the interface iterations decrease successfully, but this formulation still gives longer computation time compared to the auxiliary potential formulation together with the modified parallelization.

Table 9.6. Comparison of speed-up values obtained by the new modified domain decomposition computation and by the first parallelization method, on SGI O3000 system

SGI ORIGIN 3000	LID-DRIVEN CUBIC CAVITY (Re=1000, Δt=0.1)		
Number of domains	2	4	6
Speed-up (elapsed time) First parallelization run	1	2.30	2.40
Speed-up (elapsed time) New modification run	1	2.71	3.12
Speed-up (CPU time) First parallelization run	1	2.95	4.91
Speed-up (CPU time) New modification run	1	3.31	5.69

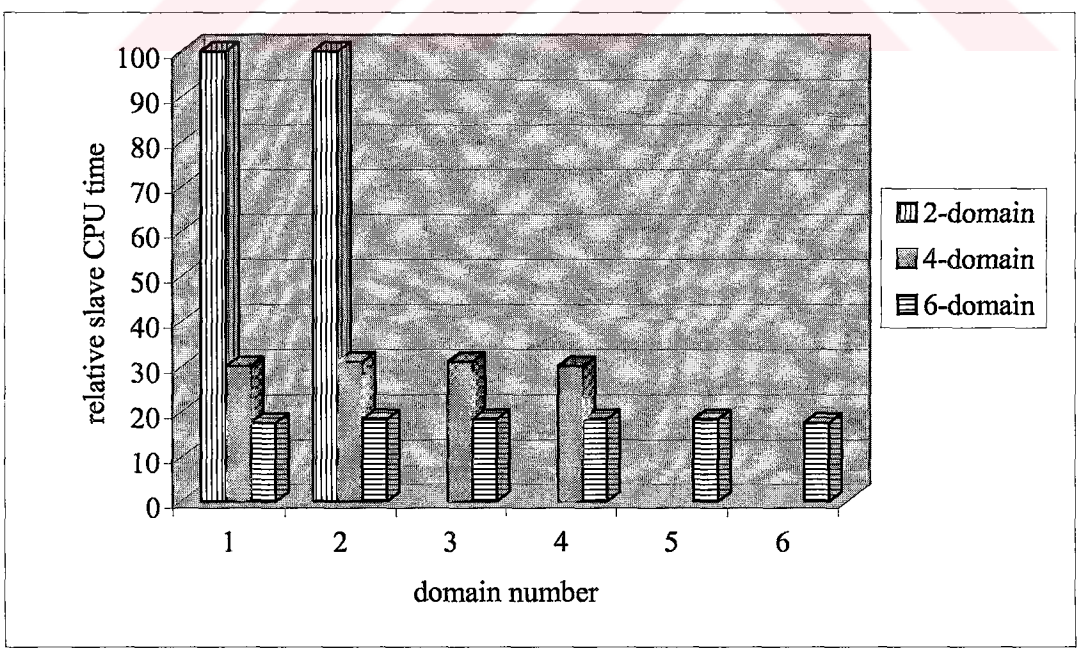


Figure 9.2. New modified domain decomposition equations: relative CPU time of each slave processor computation, SGI ORIGIN3000, total number of 25x13x25 nodes, Re=1000, Δt=0.1.

10. CONCLUSIONS

The numerical solution of the 3-D Navier-Stokes equations to study large-scale problems has gained considerable attention in recent years. The large-scale problems considered necessitate the utilization of parallel computing. In this study, new parallel implicit method is introduced for the solution of 3-D Navier -Stokes equations. It is hoped to contribute towards the development of mathematical models of parallel performance. The Finite Element Method with a modified version of the two-step fractional method is used for solution of momentum equations. The scheme used is second-order accurate in both time and space. This allows us to resolve the flow field with less number of points while taking large time steps. All the results are obtained successfully with time-step size of 0.1 carried on the computational domains with minimum grid size of 0.014. The domain decomposition is employed for the parallel solution of both momentum and the auxiliary potential equations.

The implicit method is modified with direct auxiliary potential function formulation and tested via the sequential runs, successfully. Among the four different solution methods described in Chapter 6, the direct solution of auxiliary potential formulation for the pressure gives the most efficient results considering the computational time and easy programming for the parallel implementation in this study. This direct method is employed in parallel programming.

The cubic lid-driven cavity problem is solved efficiently for Re=400, via the parallel method, on 4 x (7x13x25) grid points. Velocity distributions on the symmetry plane are compared with the solutions of reference Ku *et. al.,* [71], and *Babu and Korpela*, [72].The results agree reasonably well with those of Ku *et al.*. The Reynolds number 400 results give reasonably better agreement than those of *Babu and Korpela,* when compared to Ku *et al.* results. Although their computations are carried on fine grid with 63 x 33 x 63 nodes, ours are on much more coarse grid, with 4-domain partitioning including total number of 25x 13 x 25 nodes and with a large time-step

size of 0.1. The primary flow direction is, being from left to right, longitudinal. For Re=1000, the primary eddy at the center and longitudinal corner vortices where the top and the side walls meet are observed. Similar vortices are at the junction of bottom plate and the side walls. For Re values of the order of 400 a large primary eddy occupies the core of the flow and two secondary eddies are lodged in the downstream and upstream corners on the bottom surface. As Reynolds number increases, convection becomes increasingly dominant. The primary vortex is offset towards the geometric center of the cavity.

Three dimensional lid-driven cavity test problem with Reynolds number of 1000, with 2, 4 and 6-domain partitioning is chosen initially for calibration of the written parallel code. The results give reasonably well agreement with the benchmark solution of Ku *et al.*, [71], although the computations are carried on coarse grids with total number of 21x11x21 and 25x13x25 nodes. But, the first results obtained with the new implicit parallel method are unsuccessful in terms of the speed-up, on DEC Alpha XL266 workstations. Based on the 2-domain partitioning solution, the 4-domain solution speed-up is 1.37 as opposed to its ideal value, 2, and 6-domain solution speed-up is 0.89. Since the processors used in the parallel computation on DEC Alpha XL266 workstations are not equal, load balancing is necessitated to improve the speed-up. Most of the parallel work should be loaded on the fast processors, and each processor does not have to wait for master or other slave processors to continue its own computations.

Some parallelization methods are investigated and with the third improvement, explained in Chapter 7, the new load balance modifications in the parallel code could obtain the efficient results on the parallel workstations and the speed-up (elapsed) is increased to 2.39 with 4-domain computation and 2.60 with 6-domain computation. At all the computations in this study, the best interface iteration convergence criteria is found 10^{-5} for half-time velocity and 10^{-6} for auxiliary potential function. The results are ideal; the explicit computation result of 4-domain speed-up (elapsed time) is 2.17 in reference [51], obtained on the same processors. The speed-up values concerning the overall elapsed time is normalised with respect to the 2-domain solution. The lid-driven cubic cavity flow (3-D) can be accurately computed in 4105 seconds with 2-domain, 1719 seconds with 4-domain and 1579 seconds with the 6-

domain partitioning runs on DEC Alpha XL266 workstations, via the third parallelization method.

As the domain number increases, the elapsed and CPU time decrease but the amount of the decrease of elapsed time in 4-domain solution is much more than that of 6-domain case; this is probably related to the size of half-bandwidth difference between the partitioning cases. The larger the size of half-bandwidth, the more the data to be processed and transported. Half-bandwidth in 2-domain, 4-domain and 6-domain solutions is 184, 100 and 72, respectively, where much more decrease of half-bandwidth is obtained in 4-domain partitioning case compared to that of the 6-domain case. Parallel efficiency could be much more if 8 parallel processors could exist.

Even the speed-up (CPU time) of 4.85 is obtained via the third paralellization method in 6-domain solution for cubic cavity flow, the speed-up based on the elapsed time is not as high as the CPU speed-up; this is due to the increasing number of interface inner iterations, especially the auxiliary potential interface iterations. As the interface number increases, the amount of interface data to be processed, to get the correct Neumann conditions on the interface, increases. The increase in interface iteration causes the time delay at the parallel solution on the interface. The slave processors spends too much time before starting the computations in their sub-domains with the correct conditions on their interface boundaries, and the master processor spends too much time for post-processing of the interface data. On the other hand, the time lost at the communications between the master and slave processors increases, too. With the final work in this study, this problem is overcome and interface iterations are decreased efficiently and super-linear speed-up is obtained.

The memory requirement problem in our implicit code, for large-scale problem solution, is solved successfully by getting rid of huge size matrices; the size of the code is reduced as explained in Chapter 7. The velocity and pressure fields are obtained accurately and efficiently with 2, 4 and 6-partitionings, with total number of 25x13x25 grid points, up to 55x28x55. The results could be obtained with large time-step sizes as 0.1 or 0.05, even with minimum grid size of 0.0117. Especially

with a fine stretching near the walls, the more accurate results are observed. They agree well reasonably with the bencmark solutions. The accuracy could be even more improved using finer grid elements near the walls. The primary and secondary flows inside the three dimensional lid-driven cavity are observed in the plots of the results on different cross-sections of the symmetry part of the cavity.

Solution for the full geometry of the cubic cavity is computed removing the symmetry boundary conditions and the symmetry is checked on the symmetry plane (y=0.5 plane). The results are exactly same as the results obtained from the computations including only the symmetry part. Symmetric character of the flow is shown by the results obtained in Chapter 8.

Because of the breakdown of the DEC-Alpha cluster of workstations, the studies after Chapter 7 is continued on SGI Origin 3000 utilized with 8 processor running Unix operating system. Parallel code written in this study is tested on this new system and compared with the results obtained on DEC-Alpha cluster of workstations. The speed-up (elapsed time) obtained on SGI Origin 3000 is calculated 2.30 for 4-domain and 2.40 for 6-domain partitioning runs, based on the 2-domain result of the cubic lid-driven cavity flow with Reynolds number 1000. If the CPU time is considered, the speed-up reaches up to 2.95 with 4-domain computation and up to 4.91 with 6-domain computation.

The third parallelization method, where the load balancing is employed for the computations on DEC-Alpha cluster of workstations, is found not efficient on SGI Origin 3000 system as the first parallelization method, where the parallel work is shared equally between the master and slave processors. The reason is that, the processors are exactly equal on SGI Origin 3000 system and this necessitates equal share of work between the slave processors, including the master.

As explained in detail in Chapter 9, in this work, some parallel solutions ways are tested to decrease the interface auxiliary potential iterations. The parallel code is modified to calculate the pressure directly instead of calculating the pressure via the auxiliary potential function. The results agree well with the benchmark solutions, but it is seen that the overall computation time is increased a bit more, if the domain

decomposition technique is employed with the direct pressure formulation, instead with direct auxiliary potential function formulation.

Speed-up analysis is continued and finally it is achieved to decrease the interface iterations for the auxiliary potential, about 50 percent. The interface Neumann boundary conditions at the beginning of each time level are set equal to the converged value, which is found out at the end of the interface iterations at the previous time level, but not zero. The domain decomposition equations, explained in Chapter 5, are modified with this idea, and finally interface computations are converged more easily, the number of iterations is reduced efficiently and super-linear speed-up is obtained.

The computational time (elapsed) obtained with the modified domain decomposition equations, on 6-domain partitioning, with total number of 6 x (5x13x25) nodes, reduces from 227 seconds to 151 with time-step size of 0.1, and with 6 x (10x28x55) nodes, reduces from 9774 seconds to 7508 seconds, with time-step size 0.05 (run on SGI Origin 3000).

With this latest modification, the speed-up (elapsed time), based on the 2-domain partitioning parallel result, is increased from 2.30 (first parallelization) to 2.71 on 4-domain partitioning, and from 2.40 to 3.12 on 6-domain partitioning, on SGI Origin 3000 system. Speed-up (CPU time) is 3.31 on 4-domain and rises up to 5.69 on 6-domain partitioning computations.

In this study, the parallel, implicit code solves successfully the test problem on the simple domain with a complex flow, but the complex flow on complex domains can also be investigated. Number of domains can be increased by using different machines connected to the network and parallel efficiency and speed-up can be analyzed. Energy equation can be added to the parallel, implicit Navier-Stokes solver and heat transfer calculations can be done by the parallel computations.

REFERENCES

[1] **Ferziger, J. H., and Peric, M.**, 1996. Computational Methods for Fluid Dynamics, pp. 1-195, 297-316, Springer-Verlag, Heidelberg.

[2] **Jameson, A.**, 1996. The present status, challenges, and future developments in Computational Fluid dynamics. In Progress and Challenges in CFD Methods and Algorithms (Seville). AGARD-CP-578.

[3] **Kutler, P.**, 1984. A perspective of theoretical and applied Computational Fluid Dynamics. *AIAA Journal* 23, 328-341.

[4] **Edis, F. O.**, 1998. Efficient Finite Element Computation of Unsteady Incompressible Viscous Flows Using Pseudo-Second-Order Velocity Interpolation, *Ph.D. Thesis,* İ.T.Ü. Fen Bilimleri Enstitüsü, İstanbul.

[5] **Kwak, D.**, 1989. Computation of Viscous Incompressible Flows. No. 1989-04 in von Karman Institute for Fluid Dynamics Lecture Series.

[6] **Hirsch, C. H.**, 1990. Numerical Computation of Internal and External Flows, Vol 2, pp. 595-674, John Wiley & Sons.

[7] **Anderson, J. D.**, 1991. Fundementals of Aerodynamics, pp. 3-390, second ed., McGraw-Hill.

[8] **Gunzburger, M. D.**, 1989. Finite Element Methods for Viscous Incompressible Flows: A Guide to Theory Practice and Algorithms, first ed. Academic Press, San Diego.

[9] **Zienkiewicz, O., and Taylor, R. L.**, 1991. The Finite Element Method, Volume 2, fourt ed., McGraw-Hill.

[10] **Nakajima, K., and Kallinderis, Y.**, 1993. Comparison of finite element and finite volume methods for incompressible viscous flows. *AIAA Journal* 32, 5, 1090-1093.

[11] **Lohner, R.**, 1987. Finite elements in CFD: what lies ahead. Int. J. Numer. Methods Eng 24, 1741-1756.

[12] **Waterson, N. and Deconinck, H.**, 1996. Development of a 3D Incompressible Unstructured Navier-Stokes Solver, *Technical Annex*, VKI.

[13] **Pironneau, O.**, 1989. Finite Element Methods for Fluids, first edition, John Wiley & Sons.

[14] **Hughes, T., and Brooks, A.**, 1995. Stability analysis of mixed Finite element formulations with special mention of equal-order interpolations, *Int. J. Numer. Methods Fluids* 20, 1003-1022.

[15] **Fortin, M.**, 1981. Old and new finite elements for incompressible flows. *Int. J. Numer. Methods Fluids* 1, 347-364.

[16] **Akl, S.G.**, 1997. Parallel Computation, Models and Methods, Prentice Hall.

[17] **Freeman, T. L.**,1992. Parallel Numerical Algorithms, Prentice Hall.

[18] **Emerson, D. R., Ecer, A., Periaux, J., Satofuka, N. And Fox, P.**, 1997. Parallel Computational Fluid Dynamics, Recent Developments and Advances Using Parallel Computers, pp. V, *Parallel CFD 97 Conference*, Manchester, U.K, Elsevier.

[19] **Hauser, J., Simon, H. D., and Paap, H.G.**, 1992. Features of Architecture Independent Parallel CFD Software, *Parallel Computational Fluid Dynamics'92*, Edited by R. B. Pelz et al., North-Holland, Amsterdam.

[20] **Lin, A.**, 1990. Parallel Numerical Algorithms for Fluid Dynamics Simulation, AIAA90-0333.

[21] **Nicola, C., Pietro, G., and Paparone, L.**, 1992. Evaluation of Different Approaches to Parallel Processing for CFD, *Parallel Computational Fluid Dynamics'92*, Edited by R. B. Pelz et al., North-Holland, Amsterdam.

[22] **Nicola, C., Pietro, Puoti, V., Schiano, P., and Vaccaro, R.**, 1991. Parallel Processing for Typical CFD problems, *XI AIDAA Conference*, Forli'.

[23] **Linden, J., Lonsdale, G., Ritzdorf, H., and Schüller, A.**, 1992. Block-Structured Multigrid for the Navier-Stokes Equations: Experiences and Scalability Questions, *Parallel Computational Fluid Dynamics'92*, Edited by R. B. Pelz et al., North-Holland, Amsterdam.

[24] **Peric, M., Schafer, M., and Schreck, E.**, 1992. Numerical Simulation of Complex Fluid Flows on MIMD Computers, *Parallel Computational Fluid Dynamics'92*, Edited by R. B. Pelz et al., North-Holland, Amsterdam.

[25] **Peric, M., Schafer, M., and Schreck, E.**, 1991. Computation of Fluid Flow with a parallel multi-grid solver, *Proc. Int. Conference on 'Parallel Computational Fluid Dynamics'*, Amsterdam, Elsevier.

[26] **Hockney, R. W., and Jesshope, C. R.**, 1988. Parallel Computers: Programming and Algorithms, Adam Hilger.

[27] **Smith, D. M., and Fiddes, S. P.**, 1992. Efficient Parallelisation of Implicit and Explicit Solvers on a MIMD computer, *Parallel Computational Fluid Dynamics'92*, Edited by R. B. Pelz et al., North-Holland, Amsterdam.

[28] **Flatt, H. P., and Kennedy, K.**, 1989. Performance of Parallel Processors, *Parallel Computing*, 12, pp. 1-20.

[29] **Gropp, W. D., and Smith, E. B.**, 1990. Computational Fluid Dynamics on Parallel Processors, *Computers & Fluids*, 18, 289-304.

[30] **Nicol, D. M., and Willard, F. H.**, 1988. Problem Size, Parallel Architecture, and Optimal Speedup, *Journal of Parallel and Distributed Computing*, 5, 404-420.

[31] **Akay, H. U., Ecer, A., and Fekete, K.**, 1997. A Domain Decomposition Based Parallel Solver for Viscous Incompressible Flows, *Parallel CFD 97 Conference*, Manchester, U.K, Elsevier.

[32] **Ecer, A., Akay, H. U., Kemle, W. B., Wang, H., Ercoskun, D., and Holl, E. J.**, 1994. Parallel computation of fluid dynamics problems, *Comp. Methods Appl. Mech. Eng.*, Vol. 112, pp. 91-108.

[33] **Akay, H. U., Ecer, A., and Fekete, K.**, 1997. A Semi-Explicit Parallel Solver for Viscous Incompressible Flows, *Comp. Methods Appl. Mech. Eng.*

[34] **Akay, H. U., Blech, R., Ecer, A., Ercoskun, Z., Kemle, W. B., Quealy, A., and Williams, A.**, 1993. A Database Management System for Parallel Processing of CFD Algorithms, *Parallel Computational Fluid Dynamics'92*, Edited by R. B. Pelz et al., North-Holland, Amsterdam.

[35] **Simon, H. D.**, 1991. Partitioning of unstructured problems for parallel processing, *Computing systems in engineering*, 213, pp. 135-148.

[36] **Quagliarella, D.**, 1995. Optimal Domain Decomposition for Parallel Multiblock Flow-field Solvers using Genetic Algorithms, *Parallel Computational Fluid Dynamics: newalgorithms and applications*, pp. 215-222, Elsevier, Amsterdam.

[37] **Bucchignani, E., Diurno, W. G.**, 1997. Parallel computation of inviscid 3D flows with unstructured domain partitioning: performance on SGI-Power Challenge Supercomputer, *Parallel Computing'97 Conference*, Bonn.

[38] **Bucchignani, E., Mella, R., Schiano, P., Richelli, G.**, 1997. Comparisons of the MPI and PVM performances by using structured and unstructured CFD codes, Parallel Computational Fluid Dynamics, Recent Developments and Advances Using Parallel Computers, pp. 125-132, *Parallel CFD 97 Conference*, Manchester, U.K, Elsevier.

[39] **Lanteri, S., Nkonga, B., Piperno, S.**, 1997. Domain partitioning and message passing for the distribution of unstructured mesh calculations on MIMD platforms: application to steady and unsteady compressible flow simulations, Parallel Computational Fluid Dynamics: *Algorithms and results using advanced computers*, pp. 11-20, Elsevier, Amsterdam.

[40] **Forsey, C. R., Ierotheou, C. S., Block, U., Leatham, M.**, 1996. Parallelization and performance evaluation of the aeronautical CFD flow code ESAUNA, *Proc. HPCN Europe 1996*, pp. 599-606, Springer, Berlin.

[41] **Patankar, S. V.**, 1980. Numerical Heat Transfer and Fluid flow, Hemisphere, Washington D.C.

[42] **Roache, P. J.**, 1972. Computational Fluid Dynamics, Hermosa, New Mexico.

[43] **Zienkiewicz, O., and Taylor, R. L.**, 1989. The Finite Element Method, Volume 1, pp. 479-295, fourt ed., McGraw-Hill.

[44] **Gülçat, Ü. and Aslan, A.R.**, 1997. Accurate 3D Viscous Incompressible Flow Calculations with the FEM, *International Journal for Numerical Methods in Fluids,* 25, 985-1001.

[45] **Chorin, A. J.**, 1968. Numerical Solution of the Navier-Stokes Equations, *Math. Comput.,* vol 22., pp. 745-762.

[46] **Temam, R.**, 1977. Navier-Stokes equations, Volume 2, pp. 395-417, North-Holland, Amsterdam.

[47] **Hayashi, M., Hatanaka, K., and Kawahara, M.**, 1991. Lagrangian finite element method for free surface Navier-Stokes flow using fractional step methods, *Int. J. Numer. Methods Fluids* 13, pp. 805-840.

[48] **Despotis, G., and Tsangaris, S.**, 1995. Fractional step method for solution of incompressible Navier-Stokes equations on unstructured triangular meshes, *Int. J. Numer. Methods Fluids* 20, pp. 1273-1288.

[49] **Türek, S.**, 1996. A comperative study of time-stepping techniques for the incompressible Navier-Stokes equations: from fully implicit non-linear schemes to semi-implicit projection methods. Int. J. Numer. Methods Fluids 22, pp. 987-1011.

[50] **Marx, Y.**, 1994. Time integration schemes for the unsteady incompressible Navier-Stokes equations, J. Comput. Phys. 112, pp. 182-209.

[51] **Aslan, A.R., Edis, F.O., Gülçat, Ü., and Mısırlıoğlu, A.**, 1998. Accurate Incompressible N-S Solution on Cluster of Workstations, *Parallel CFD'98,* Hsinchu, TAIWAN, May 11-14.

[52] **Aslan, A.R., Gülçat, Ü., and Mısırlıoğlu, A.**, 1995. A PCG/EBE iteration for high order and fast solution of 3-D Navier-Stokes equations, Progress and Challenges in CFD Methods and Algorithms (Seville), AGARD-CP-578.

[53] **Gülçat, Ü., Üstoğlu Ünal, V.**, 2000. Accurate Implicit Solution of 3D Navier-Stokes Equations on Cluster of Workstations, *Pro. of Parallel CFD 2000 Conference*, Trondheim, Norway, Elsevier.

[54] **Gülçat, Ü., Aslan, A. R., Üstoğlu Ünal, V.**, 2000. Accurate Implicit Solution of 3D Navier-Stokes Equations, *ECCOMAS 2000 Conference*, Barcelona.

[55] **Gülçat, Ü., Üstoğlu Ünal, V.**, 2001. 'Üç boyutlu Navier-Stokes denkleminin paralel, hassas kapalı sayısal çözümü', *XII. Ulusal Mekanik Kongresi*, Konya.

[56] **Mizukami, A., and Tsuchiya, M.**, 1984. A Finite Elelement Method For the 3-D Non-steady Navier Stokes Equations, *International Journal for Numerical Methods in Fluids,* vol 4., pp. 349-357.

[57] **Utnes, T., and Ren, G.**, 1995. Finite element modelling of the incompressible Navier-Stokes equations, Comp. Fluid Dyn., 4, pp. 41-55.

[58] **Finlayson, B. A.**, 1972. The Method of Weighted Residuals and Variational Principles, Academic Press..

[59] **Finlayson, B.A.**, 1982. Existence of Variational Principles for the Navier-Stokes Equations, *The Physics of Fluids,* vol 15., No 6., June.

[60] **Galerkin, B.G.**, 1915. Rods and Plates, Series occuring in various questions concerning the Elastic Equilibrium of Rods and Plates, vol 19., pp. 897-908.

[61] **Ralston, A.**, 1965. A First Course in Numerical Analysis, McGraw-Hill.

[62] **Fox, L.**, 1965. An Introduction to Numerical Linear Algebra, Oxford University Press.

[63] **Meyer, C.**, 1975. Special problems related to linear equation solvers, *J. Struct. Div. ASCE*,101,969-90.

[64] **Glowinski, R., and Periaux, J.**, 1982. Domain Decomposition Methods for Nonlinear Problems in Fluid Dynamics, *Research Report* 147, Inria, FRANCE.

[65] **Dinh, Q.V., Ecer, A., Gülçat, Ü., Glowinski, R., and Periaux, J.**, 1992. Concurrent Solutions of Elliptic Problems via Domain Decomposition, Applications to Fluid Dynamics, *Parallel CFD 92,* Rutgers University, May 18-20.

[66] **Glowinski, R., Pan, T.W., and Periaux, J.**, 1995. A One Shot Domain Decomposition/Fictitious Domain Method for the Solution of Elliptic Equations, *Parallel CFD 95,* Elsevier.

[67] **Lin, C.A., Ecer, A., Satofuka, N., Fox, P., and Periaux, J.**, 1999. Development and Applications of Parallel Technology, *Parallel Computational Fluid Dynamics,* Indiana, U.S.A.

[68] **Hirsch, C. H.**, 1990. Numerical Computation of Internal and External Flows, Vol 2, pp. 670-675, John Wiley & Sons.

[69] **Ghia, U., Ghia, K.N., and Shin, C.T.**, 1982. High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method, *J. of Computational Phys.,* Vol 48, pp 387-411.

[70] **Üstoğlu, V.**, 1997. Implementation and Validation of Residual Distribution Schemes for Multidimensional Incompressible Viscous Flow and Heat Transfer Calculations, *Diploma Course thesis*, Von Karman Institute, Belgium.

[71] **Ku, H. C., Hirsh, R., S., and Taylor T., D.,**, 1987. *Journal of Computational Physics,* Vol 70, 439.

[72] **Babu, V., and Korpela, S., A.,** 1994. Numerical Solution of the Incompressible Three-Dimensional Navier-Stokes Equations, *Computor Fluids,* Vol. 23, No. 5, pp. 675-691.

APPENDIX A. DIRECT SOLUTION: UN-SYMMETRIC ACTIVE COLUMN PROFILE SOLVER

```
C       parameter nband: half-bandwidth, nnode: number of nodes
        A(nnode x nband), C(nnode x nband), JDIAG(nnode)
C       solve momentum equation at half-time step
        afac=.true.
        back=.true.
        call unsym(a,cc,vnh,jdiag,neq,afac,back)
        afac=.false.
        back=.true.
        call unsym(a,cc,unh,jdiag,neq,afac,back)
        call unsym(a,cc,wnh,jdiag,neq,afac,back)
C
        SUBROUTINE UNSYM(A,C,B,JDIAG,NEQ,AFAC,BACK)
        LOGICAL AFAC, BACK
        DIMENSION A(1),B(1),JDIAG(1),C(1)
C
C       UN-SYMMETRIC ACTIVE COLUMN PROFILE SOLVER
C
C       A(NAD)  UPPER TRIANGULAR COLUMNS
C       B(NEQ)  RIGHT-HANDSIDE
C       C(NAD)  LOWER TRIANGULAR ROWS
C       JDIAG(NEQ) POINTER ARRAY TO DETERMINE IN A OR C OF
C       DIAGONAL PIVOTS
C       NEQ NUMBER OF EQUATIONS = nnode number of nodes
C       NAD LENGTH OF A OR C ARRAYS = JDIAG(NEQ)
C       AFAC: IF TRUE, TRIANGULAR DECOMPOSITION PERFORMED
C       BACK: IF TRUE, FORWARD REDUCTION AND
C       BACK SUBSTITUTION PERFORMED
C       FACTOR A
        JR=0
        DO 300 J=1,NEQ
        JD=JDIAG(J)
        JH=JD-JR
        IF(JH.LE.1)GO TO 300
        IS=J+1-JH
        IE=J-1
        IF(.NOT.AFAC)GO TO 250
```

```
      K=JR+1
      ID=0
C
C     REDUCE ALL EQUATIONS EXCEPT DIAGONAL
C
      DO 200 I=IS,IE
      IR=ID
      ID= JDIAG(I)
      IH=MIN(ID-IR-1,I-IS)
      IF(IH.EQ.0) GO TO 150
      A(K)=A(K)-DOT(A(K-IH),C(ID-IH),IH)
      C(K)=C(K)-DOT(C(K-IH),A(ID-IH),IH)
150   IF(A(ID).NE.0.0) C(K)=C(K)/A(ID)
200   K=K+1
C
C     REDUCE DIAGONAL TERM
C
      A(JD)=A(JD)-DOT(A(JR+1),C(JR+1),JH-1)
C
C     FORWARD REDUCE THE RHS
C
250   IF(BACK) B(J)=B(J)-DOT(C(JR+1),B(IS),JH-1)
300   JR=JD
      IF(.NOT.BACK)RETURN
C
C     BACK SUBSTITION
      J=NEQ
      JD=JDIAG(J)
 500  IF(A(JD).NE.0.0)B(J)=B(J)/A(JD)
      D=B(J)
      J=J-1
      IF(J.LE.0)RETURN
      JR=JDIAG(J)
      IF(JD-JR.LE.1) GO TO 700
      IS=J-JD+JR+2
      K=JR-IS+1
      DO 600 I=IS,J
 600  B(I)=B(I)-A(I+K)*D
 700  JD=JR
      GO TO 500
      END
      FUNCTION DOT(A,B,N)
      DIMENSION A(n),B(n)
      DOT=0.
      DO I=1,N
      DOT=DOT+A(I)*B(I)
      ENDDO
      RETURN
      END
```

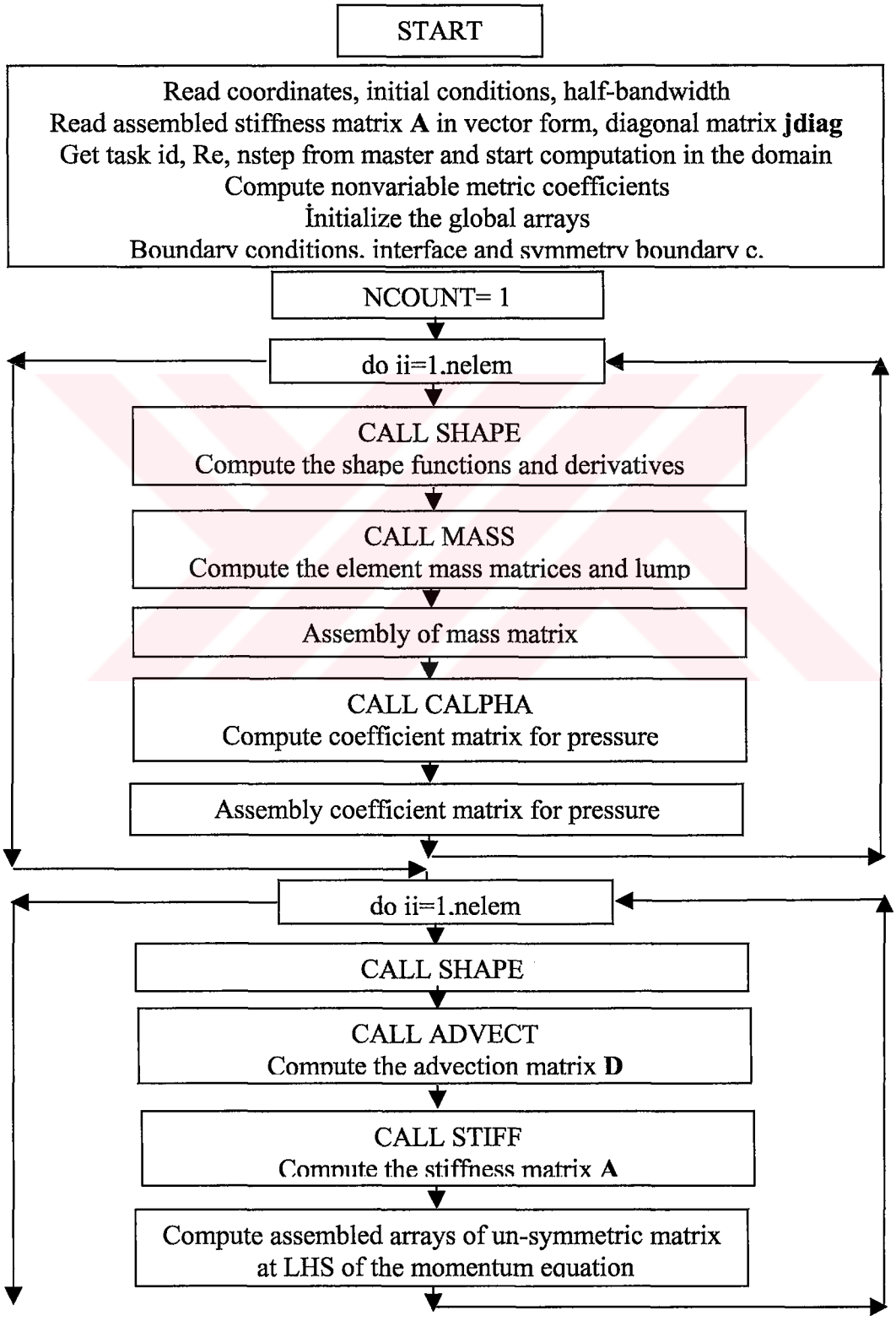
APPENDIX B. DIRECT SOLUTION: SYMMETRIC PROFILE SOLVER

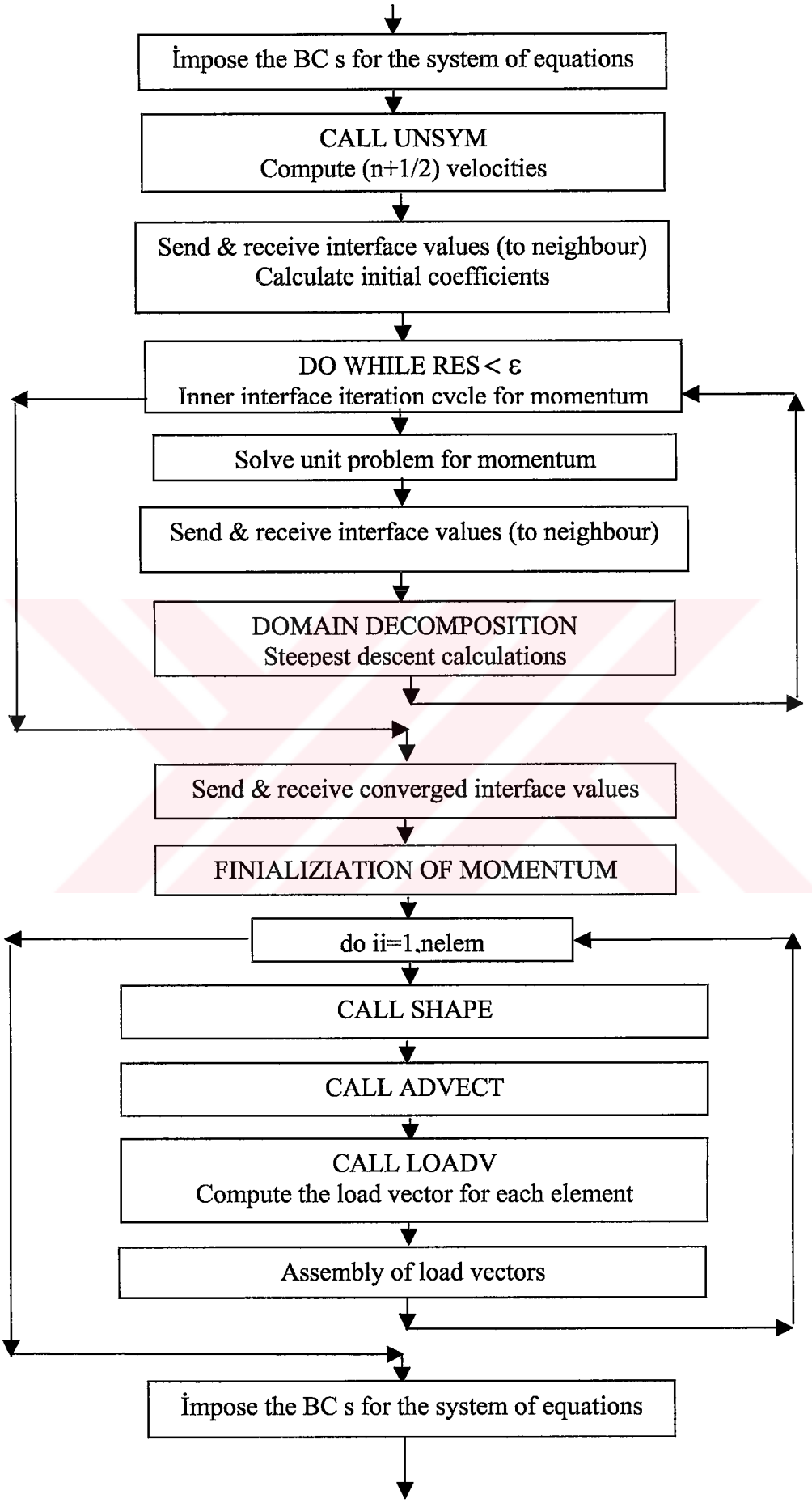
```
      parameter nband: half-bandwidth, nnode: number of nodes
      A1(nnode x nband), JDIAG(nnode)
C     solve auxiliary potential equation at half-time step
C     r : load vector, RHS of auxiliary potantial equation
C     ss : stiffness matrix
      afac=.false.
      IF(NCOUNT.EQ.1)THEN
      afac=.true
      call yersym(a1,ss,neq,nband)
      ENDIF
       back=.true.
C
      call sym(a1,r,jdiag,neq,afac,back)
C
      SUBROUTINE YERSYM(a1,ss,neq,nband)
      dimension a1(1),ss(neq,1)
      ll=nband
      ne=1
      ii=0
 55   ir=ne
      jn=ne+1
      if(ne.gt.nband)ir=nband
      if(ne.gt.ll)jn=ll+1
      do 130 i=1,ir
      ii=ii+1
      in=ne-ir+i
      jn=jn-1
C     if(in.gt.nband)jn=jn-1
      a1(ii)=ss(in,jn)
 130  continue
      jd(ne)=ii
      ne=ne+1
      if(ne.le.neq)go to 55
      return
      end
```

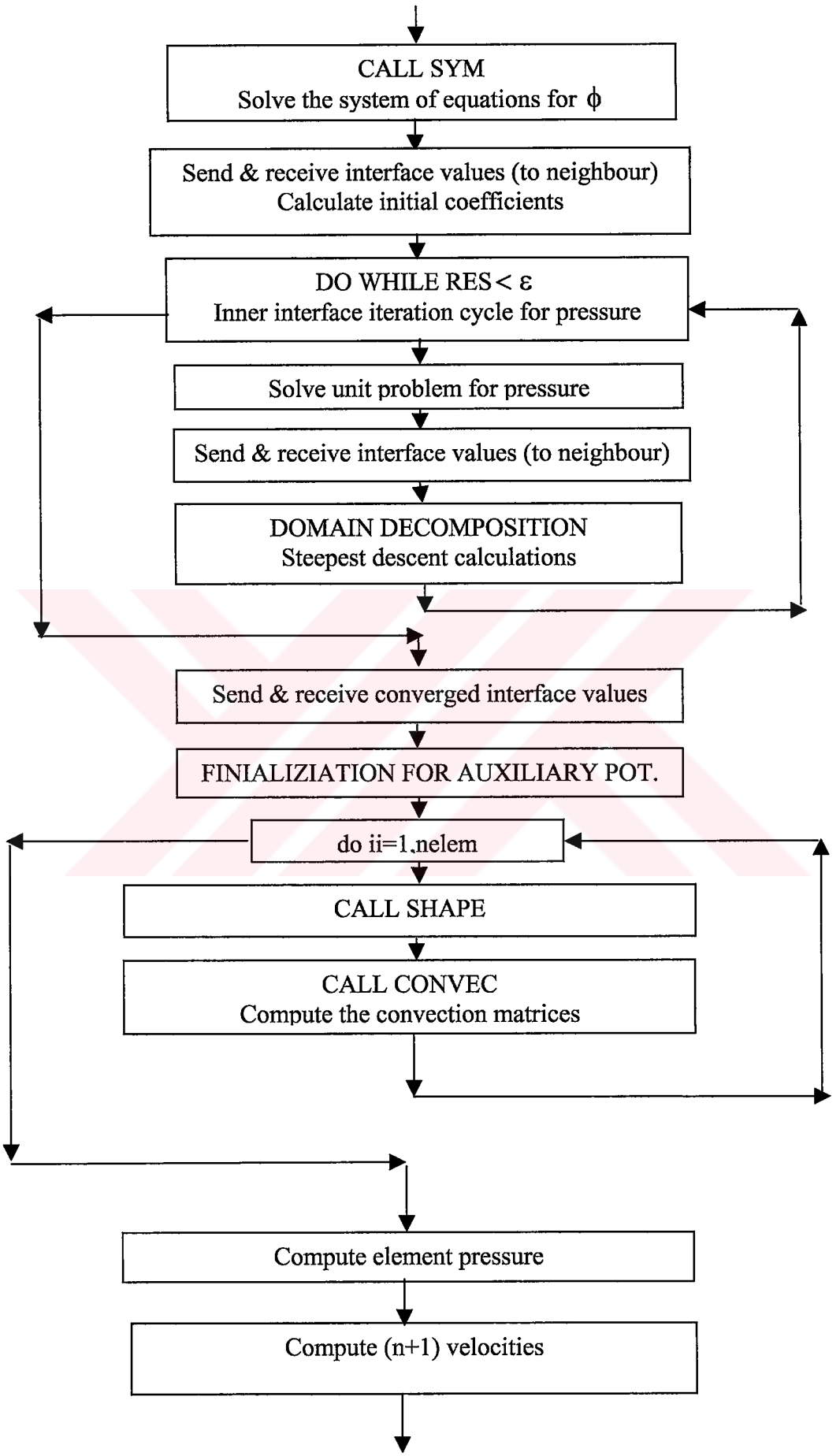
```
      SUBROUTINE SYM(A,B,JDIAG,NEQ,AFAC,BACK)
      LOGICAL AFAC, BACK
      COMMON/ENGYS/AENGY
      DIMENSION A(1),B(1),JDIAG(1)
C
C     SYMMETRIC ACTIVE COLUMN PROFILE SOLVER
C
C     AFAC: IF TRUE, TRIANGULAR DECOMPOSITION PERFORMED
C
C     BACK: IF TRUE, FORWARD REDUCTION AND
C     BACK SUBSTITUTION PERFORMED
C
      AENGY=0.0
      JR=0
      DO 600 J=1,NEQ
      JD=JDIAG(J)
      JH=JD-JR
      IS=J-JH+2
      IF(JH-2) 600,300,100
  100 IF(.NOT.AFAC) GO TO 502
      IE=J-1
      K=JR+2
      ID=JDIAG(IS-1)
      DO 200 I=IS,IE
      IR=ID
      ID=JDIAG(I)
      IH=MIN(ID-IR-1,I-IS+1)
      IF(IH.GT.0) A(K)=A(K)-DOT1(A(K-IH),A(ID-IH),IH)
  200 K=K+1
  300 IF(.NOT.AFAC) GO TO 502
      IR=JR+1
      IE=JD-1
      K=J-JD
      DO 400 I=IR,IE
      ID=JDIAG(K+i)
      IF(A(ID).EQ.0.0) GO TO 400
      D=A(I)
      A(I)=A(I)/A(ID)
      A(JD)=A(JD)-D*A(I)
  400 CONTINUE
  502 IF(BACK) B(J)=B(J)-DOT1(A(JR+1),B(IS-1),JH-1)
  600 JR=JD
      IF(.NOT.BACK)RETURN
      DO 700 I=1,NEQ
      ID=JDIAG(I)
      IF(A(ID).NE.0.0) B(I)=B(I)/A(ID)
  700 AENGY=AENGY+B(I)*B(I)*A(ID)
      J=NEQ
      JD=JDIAG(J)
  800 D=B(J)
```

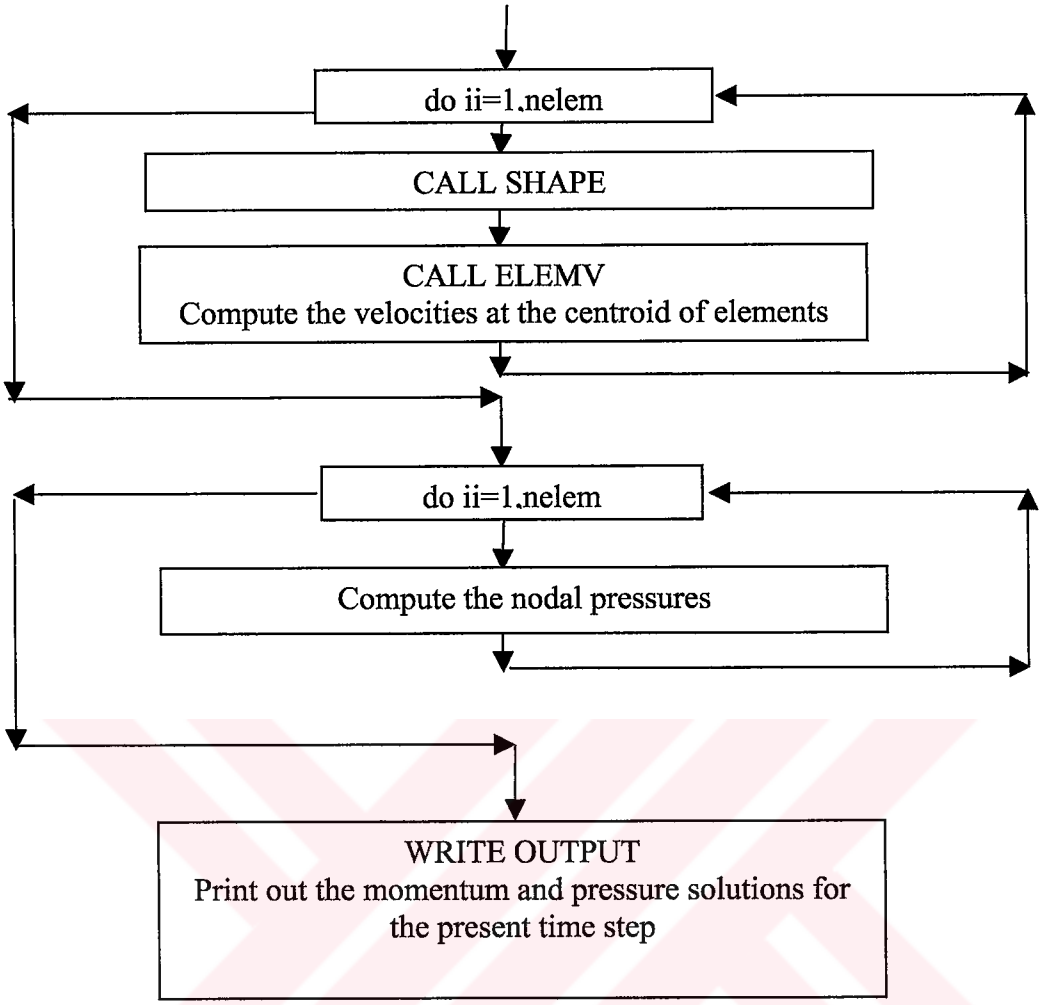
```
      J=J-1
      IF(J.LE.0)RETURN
      JR=JDIAG(J)
      IF(JD-JR.LE.1) GO TO 1000
      IS=J-JD+JR+2
      K=JR-IS+1
      DO 900 I=IS,J
  900 B(I)=B(I)-A(I+K)*D
 1000 JD=JR
      GO TO 800
      END
C
      FUNCTION DOT1(A,B,N)
      DIMENSION A(n),B(n)
      DOT1=0.0
      DO I=1,N
      DOT1=DOT1+A(I)*B(I)
      ENDDO
C
      RETURN
      END
```

APPENDIX C. FLOW CHART AT ONE TIME-STEP LEVEL




İmpose the BC s for the system of equations
CALL UNSYM
Compute (n+1/2) velocities
Send & receive interface values (to neighbour)
Calculate initial coefficients
DO WHILE RES < ε
Inner interface iteration cycle for momentum
Solve unit problem for momentum
Send & receive interface values (to neighbour)
DOMAIN DECOMPOSITION
Steepest descent calculations
Send & receive converged interface values
FINIALIZIATION OF MOMENTUM
do ii=1,nelem
CALL SHAPE
CALL ADVECT
CALL LOADV
Compute the load vector for each element
Assembly of load vectors
İmpose the BC s for the system of equations


CALL SYM
Solve the system of equations for ϕ
Send & receive interface values (to neighbour)
Calculate initial coefficients
DO WHILE RES < ε
Inner interface iteration cycle for pressure
Solve unit problem for pressure
Send & receive interface values (to neighbour)
DOMAIN DECOMPOSITION
Steepest descent calculations
Send & receive converged interface values
FINIALIZIATION FOR AUXILIARY POT.
do ii=1,nelem
CALL SHAPE
CALL CONVEC
Compute the convection matrices
Compute element pressure
Compute (n+1) velocities


do ii=1,nelem
CALL SHAPE
CALL ELEMV
Compute the velocities at the centroid of elements
do ii=1,nelem
Compute the nodal pressures
WRITE OUTPUT
Print out the momentum and pressure solutions for the present time step

VITA

Vildan Üstoğlu Ünal was born in Konya on September 26, 1971. She gratuated from Özel Ortadoğu Lisesi in 1989, and attended the Marmara University, Department of Physics, graduating in 1993 as honor student. She has been research assistant in the same university for 3 years and in 1996, she obtained MSc degree in Marmara University Institute of Science and Letters-Physics Department. She received NATO Fellowship and graduated from the diploma course at von Karman Institute for Fluid Dynamics in 1997. Same year she began the doctoral studies at I.T.U Institute of Science and Technology-Astronautical Engineering Department, and to work as a research assistant at Yeditepe University-Physics Department. Ph.D. thesis is submitted by her in June 2002.