國　立　交　通　大　學

土木工程學系

碩士論文

GPU 加速數模於二維不可壓縮穴流之研究

GPU accelerated simulations of two-dimensional incompressible cavity flow

研究生:謝東洲

指導教授:葉克家 博士

中華民國 一零二年二月

GPU 加速數模於二維不可壓縮穴流之研究

學生：謝東洲　　指導教授：葉克家

國立交通大學土木工程學系

摘要

圖形處理器(Graphic Processing Unit,GPU) 的開發起初源自於處理電腦遊戲大量貼圖運算，現今透過計算統一架構(Compute Unified Device Architecture,CUDA) 能夠有效的運用其高度計算能力、儲存器帶寬於科學計算方面。在水利方面所面臨的大量計算問題，如集水區淹水演算、三維水理演算及三維動床演算等，數據規模大小已經達到 TB 甚至於 PB 量級，因此對計算效能構成了嚴峻的挑戰。本研究藉由 GPU 以有限差分法求解二維穩態不可壓縮穴流，評估 GPU 加速於數值模擬之效益。藉由改變穴流長寬比與網格大小，得知網格數量越高，越有平行運算的必要，本研究 GPU 採用 nVidia GeForce GTX 480，CPU 方面選用 Intel® Core™2 Duo Processor E7400 與 AMD Athlon II X4 635，在長寬比為 7，網格點數達 257×1793 時，對於 intel 之 CPU 有 33 倍加速效果，AMD 之 CPU 則有 44 倍加速成效。

關鍵字：GPU、CUDA、穴流、有限差分法

GPU accelerated simulations of two-dimensional incompressible cavity flow

Student: Tung-Chou Hsieh **Advisor: Keh-Chia Yeh**

Institute of Civil Engineering

National Chiao Tung University

Abstract

The development of Graphic Processing Unit (GPU) originated from processing a great deal of mapping operation in computer games. Nowadays, GPU can apply its strong computing power and bandwidth of storage effectively to science computation by using Compute Unified Device Architecture (CUDA). There are a large amount of calculational problems we will face. For instance, model for watershed inundation, three-dimensional hydraulic model, three-dimensional mobile-bed model etc. The data size above has reached to TB even to PB, and it yields a rigorous challenge to computing efficiency. This study takes GPU combined with finite difference method to solve two-dimensional steady incompressible cavity flow, and evaluates the beneficial result of numerical simulation accelerated. By changing length of cavity flow and size of grid, we find that the more grid number, the more necessary for parallel processing. This study takes nVidia GeForce GTX 480 in GPU, and Intel® Core™2 Duo Processor E7400 and AMD Athlon II X4 635 in CPU. When the aspect ratio is 7 and grid number reaches to 257×1793, there are 33 times acceleration effect in Intel's CPU and 44 times acceleration effect in AMD's CPU.

Key word : GPU, CUDA, Cavity flow, Finite difference scheme

誌謝

終於可以確定自己能夠畢業的現在，也是終於完成雙學位的現在，我座在書桌前心中充滿感激與感動譜出這篇謝誌，心情是如此的五味雜陳。

研究所期間承蒙恩師 葉教授克家於就學期間的悉心指導與諄諄教誨，使本論文得以順利完成。研究所三年半期間，恩師對於做事方法與態度、獨力解決問題能力之培養、邏輯思考的訓練等，熱心指導讓學生受益良多。謹此獻上最誠摯的感謝。

感謝口試委員國立成功大學蔡教授長泰、國立台灣大學許教授銘熙及國網中心蔡博士惠峰細心斧正與建議，使本論文更臻完善；此外，感謝應數系吳教授金典於課業上的教導與啟發，使學生受益無窮。

研究所期間感謝仲達學長、仁凱學長與柏傑學長於研究過程中的提點與協助。感謝研究室的同伴，紹唐、唯泰、彥瑜，這兩年來的包容。感謝研究室的學弟妹，因為有你們，研究室充滿歡笑，這三年半來若缺少了各位將失色不少。

最後，衷心感謝的是含辛茹苦、撫育我成長的父母，因為你們的栽培與鼓勵下，使我在放棄夢想之際，又燃起了我不能認輸的念頭。終於在 2013 年 2 月 6 號完成了我雙主修學位的夢想。謹將此份榮耀與喜悅獻給關心我的家人、師長、同學及所有朋友們。

目錄

表目錄

圖目錄

符號說明

H:深度

i,j:網格空間座標

L:特性長度

n:網格時間座標

p:流場壓力

Re:雷諾數

t:時間座標

U:速度

u:x 方向之速度

v:y 方向之速度

w:寬度

Δt:時間間格

Δx:x 軸向空間間格

Δy:y 軸向空間間格

ρ:流體密度

μ:流體黏滯係數

Ψ:流線函數

Ω:渦度

第一章 緒論

1.1 前言

二十世紀中期開始，隨著計算機的發展，一種介於理論和實驗的研究方法即數值計算模擬，被提出運用。此種方法是運用理論解析的方式將欲解的方程式和定義域離散化，架構一計算格式，透過反覆的迭代運算，以求得方程式的近似解。以往所探討的問題大多侷限於簡單的模型問題或受限於複雜的實驗方法與成本，透過數值模擬演算將不再受限於此，現今數值模擬已經成為科學家與工程師不可或缺的方法之一。

計算流體力學的發展約始於 1950 年代，而在 1970 年代前受限於傳統電腦的運算速度以及記憶體部分，僅能處理簡單的問題。到了 1976 年具有百萬個浮點運算能力的 CRAY　IS 問世後，開始了超級電腦運算時代。到了 90 年代初期，各種結構的平行電腦開始問世，隨 CPU 由單核發展至多核，可透過多線程編程在多個 CPU 核心內實現所謂的線程平行計算，目前常見的平行語言有 OpenMP 和 Intel 的 TBB (thread building block)。在 2007 年 NVIDIA 發表了 CUDA，提倡以 GPU 進行數值平行演算。

在水利工程方面所面臨的大量計算問題，如集水區之淹水演算、三維水理演算及三維動床演算等等，數據規模大小已經達到 TB 甚至於 PB 量級，因此對計算效能構成了嚴峻的挑戰。

在計算流體力學中，穴流之模擬已經被視為測試數值方法的標準檢驗。經由穴流模擬可以測試數值方法之準確性與特性外，亦可評估數值方法是否可進一步應用於複雜流場之計算。本研究運用穴流渦度、流線及流場模擬結果，進一步評估 GPU 運用於計算流體力學之效益。

1.2 文獻回顧

1.2.1 穴流之文獻回顧

自然界中許多現象均是以偏微分方程的型式來描述，然而這些偏微分方程多半難以求其解析解，必須透過進一步的近似假設來逼近其解，因而有了數值方法。隨著時代的演變、電腦科技的進步下，這些原本需要不斷反覆迭代求解計算的過程，透過電腦計算效率的提升而得以解套，進而演變成為數值模擬。

數值模擬與傳統的理論分析、實驗研究三者組成了研究流體流動問題的完整體系，如圖 1-1 所示。

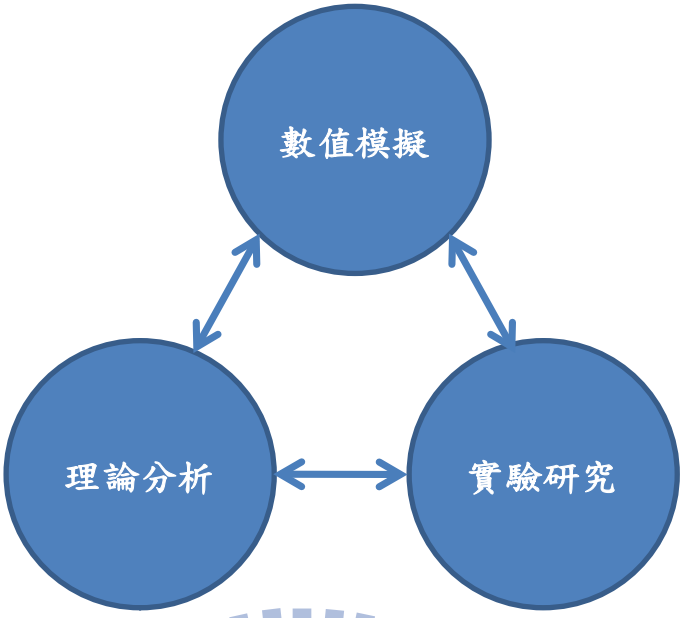


圖 1-1 流體問題解決體系圖

1. 理論分析

理論分析的優點在於所得結果具有普遍性，各種影響因素較為清晰可見，為指導實驗研究以及驗證新的數值模擬方法的理論基礎，但由於其理論過於複雜，往往需針對計算的對象進行抽象和簡化的動作才有可能求得理論解。對於非線性的情況下，只有少數的流動情況可求出解析結果。

2. 實驗研究

在運用實驗法所量測到的結果反映了問題的真實情況，為理論分析以及數值模擬的基礎，但實驗往往會受到模型尺寸的限制、流場的擾動，以及量測精度的限制，亦會因為問題太過於複雜，而很難透過實驗法來求得結果。此外實驗法往往會受到經費、人力、物力之限制，及觀測周期等造成極大的困難。

二維穴流實驗最早是由 Pan and Acrivos (1967)對 Batchelor (1956)理論

適用性之驗證，在方形體的幾何形狀比(aspect ratio)上變化以探討流場變化情形。

Koseff and Street (1984)運用染料追蹤的視覺化技術，對三維穴流做了一連串實驗，他們在三維頂部驅動封閉拉穴場的實驗觀察，發現Reynolds數介於6000至8000時，流場首次展現紊流之結構。

3. 數值模擬

數值模擬的發展彌補了實驗法與理論分析不足之處，由於電腦的蓬勃發展，可透過數學理論構築出問題的模型，類似在電腦上進行所謂的物理實驗。可以透過電腦螢幕看到流場的各種情形，如波的運動、強度，渦的產生與傳播、表面的壓力分佈等。

Burggraf (1966)首次使用渦度-流線函數方程式法(vorticity-stream function formulations method)，其雷諾數最高僅能為400。至此之後開始發展各種數值方法來探討各類穴流問題。

Benjamin 與 Denny 於1979年以渦度-流線函數方程式法(vorticity-stream function formulations method)模擬二維封閉拉穴場，並以多重網格法(multigrid method)為輔，將數值模擬計算的Reynolds數提高到 10^4，發現當雷諾數高於一特定值時，二維流場的主要組成為下列結構：位於幾何中心的主漩渦（primary eddy)，與3個位於角落的次漩渦（secondary eddy)。

Ghia et al.在1982年提出以多重網格法求解渦度-流線函數方程式法，解決高雷諾數之不可壓縮流問題，其雷諾數可達 10^4。運用多重網格法可以有效加快收斂速度，也由於其精確的計算，明顯發現主旋流與上下游次旋流的強度及大小，其模擬結果最常被採用於其他數值方法之比較基準，堪稱穴流數值模擬之典範。如Ramaswamy et al. (1992)、Spotz (1998)、Zhang (2003)等。

Schreiber和Keller在1983年提出了方穴流有效的數值方法。透過LU-factorization分解中樞，在有效寬變 量的係數矩陣下，試圖改善現有的方法，以減少所需的時間和空間來解決線性系統。

Davis (1983) 使用二階中央差分求解渦度流線方程式，提出二維穴流之標準檢驗解。Chen and Chen (1984) 首次使用有限解析法模擬穴流。

Prasad & Koseff (1989)、Koseff et al. (1983)、Koseff & Street (1984)則對長寬比為1的幾何尺寸，做了實驗上的研究，而Freitas et al. (1985)以數值方法對同樣為長寬比為1的正方型進行研究。Aidun et al. (1991) 和Benson & Aidun (1992)則是對覆蓋沉積演變過程引起研究動機，因而著手研究探討不同長寬比下的矩形流場。

Goodrich et al. (1990) & Shen (1991) 應用數值方法分析流場隨時間的轉變過程，但是此研究結果未能從實驗上得到驗證。但是隨雷諾數的增加，原流場對於主要渦漩軸(vortex axis)的對稱將受到破壞，將衍伸出許多不穩

定性問題，在這些不穩定的現象中，最重要的是離心不穩定，在離心不穩定的狀況下流場將產生所謂的Görtler-like vortices。Ramanan & Homsy (1994) 對於基本的長寬比為1的二維流場進行線性穩定度分析，發現此流場的穩定度臨界雷諾數為594 ，而三維固定的長波長的特徵向量則是造成此不穩定的主因。

Liao (1992) 推導出一種在二維穩態的 Navier-Stokes 方程式流函數的高階迭代固定邊界的解決方法。Li et al. (1995) 年提出應用在穩定不可壓縮 Navier-Stokes 方程式的緊密四階差分法。藉由一種新型的 genuine compactness 方式，研發出四階有限差分算法的獨立時間Navier-Stokes 方程式，將流線方程式依照緊密的九個點進行離散求解。

Kuhloann et al. (1997)將單板移動的問題延伸到雙板均移動的問題上。針對不同的長寬比雙板以反向相同速度及雙板不同的移動速度，對流場進行實驗與二維數值分析研究。在二維的流場當中發現了存在有多重解，例如在長寬比為 1.96 時有所謂的 two-vortex 流場和貓眼 (cat's-eye)流場，並以壁邊剪力作為參數繪出流場分歧圖，其發現流場在 Re = 427.5 會急遽的往回走，並在 Re = 234 再度向前進行，因此發現原有的二維流場並非唯一解。此外，隨著長寬比的改變，在方形體中的流場會隨時間轉變為不同的流場型態。Albensoeder et al. (2001) 探討在不同的長寬比及兩個移動的壁面間不同雷諾數下存在的流場狀況，並應用連續法來分析這些流場狀態之

存在範圍及其間的流場分歧變化情況,並準確地預測出流場的分歧點所在。在特定的長寬比下，當兩邊移動的平板以反向等速移動時，有七種不同的流場型態被預測出來，其中三種流場對稱於方形體中心，其餘四種則是非對稱的流況。而當雙面移動的平板以同向等速移動時，可找到五種不同流場型態，其中僅有一種是對稱的流況。

Cheng (2006) 使用晶格波茲曼法（lattice Boltzmann method）計算不可壓縮流場，以計算不同長寬比穴流流場變化為主。Patil (2006) 亦使用晶格波茲曼法，考慮可壓縮流場下，模擬不同長寬比穴流流場變化。

1.2.2 GPU之文獻回顧

隨時代的進步，所面臨的問題也越來越複雜化，如全球氣候的準確預報、淹水模擬、三維水理模式及三維動床模式等，數據的規模已經達到 TB 甚至 PB 量級，因此在計算速度上面臨了嚴峻的考驗。在 1976 年第一部超級電腦問世後，才開始以電腦求解複雜度較高的問題，隨電腦科技的進步，運算速度大幅提升後，得以求解規模更加龐大的問題，在 2007 年 NVIDIA 發表了以 GPU 做為計算單元的程式架構 CUDA，其單一核心下運算速度較 CPU 低落許多，但可透過其數十至數百個眾多計算核心進行平行演算，藉此彌補單一核心運算能力的不足，各時期 CPU 與 GPU 峰值浮點計算能力比較如圖 1-2 所示。

隨著資料數據達到 TB 甚至 PB 量級，在資料彼此交換傳遞方面效率也

是一個重要的問題，所幸在 GPU 在設計架構上也考慮到了此項問題，在顯示卡中的 PCB 板中可以直接焊接頻率較高的記憶體顆粒，並且 GPU 內中的記憶體控制器數量相對 CPU 多出許多，使得資料傳輸效率上亦有明顯提升，各時期 CPU 與 GPU 記憶體頻寬比較如圖 1-3 所示。

在求解數據量龐大的問題，僅能透過超級電腦運算進行求解，並逐漸發展出了 CUDA、雲端運算等，然而在這些產品中，GPU 有一項特點便是「成本較低」，因此在工程及學術界開始有許多人力投入了 CUDA 程式的撰寫。

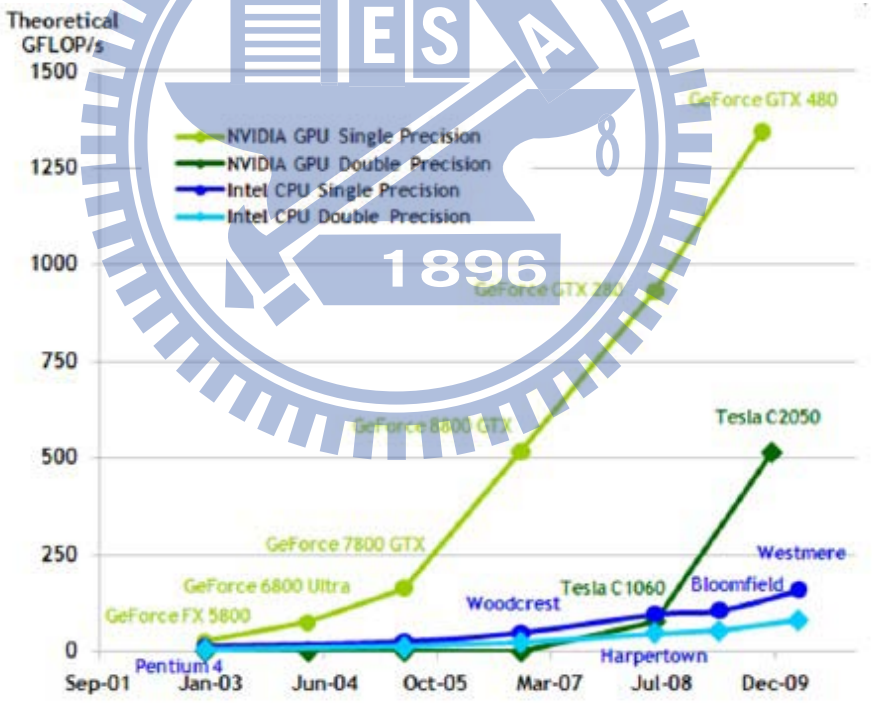


圖 1- 2 各時期 CPU 與 GPU 峰值浮點計算能力比較

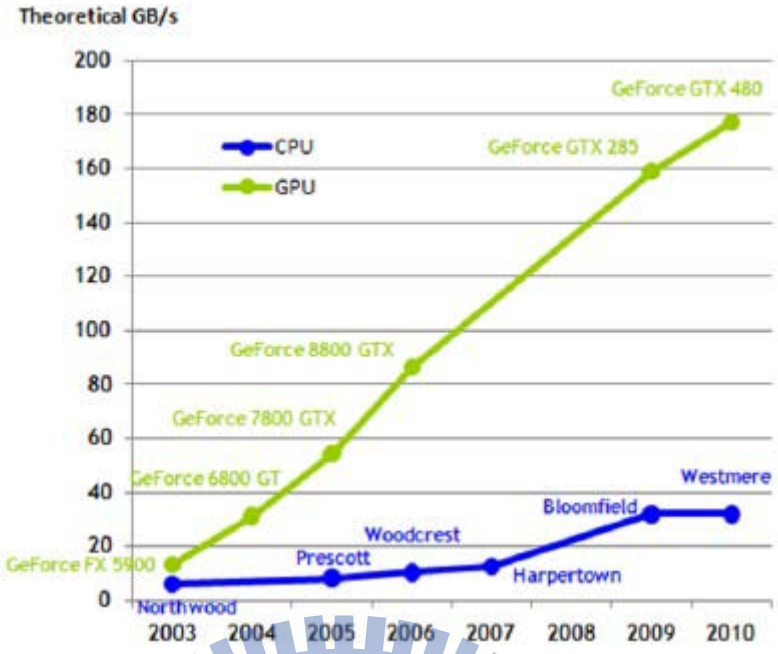


圖 1- 3 各時期 CPU 與 GPU 記憶體頻寬比較

Brandvik & Pulla (2008) 透過 GPU 求解二維及三維的 Euler 方程式，在 CPU 部分透過 Fortran 語言進行撰寫，其結果在二維部分加速了 29 倍，三維部分加速了 16 倍。Egloff & Gmbh (2010) 介紹了如何使用 GPU 求解一維偏微分方程，在文中詳細的說明計算流程以及如何最佳化求解器。

Tölke (2008) 透過 GPU 加速二維晶格波茲曼法(2D-lattice Boltzmann)，並以流體流經一個多孔介質進行加速比較，加速成果大約 10 倍。 Hérault (2008) 透過 GPU 架構 SPH 模式(smoothed particle hydrodynamics model)，其加速可達 23 倍。

Thibault and Senocak (2008) 透過 GPU 串聯，求解三維方形穴流，在 AMD Opteron (8216) 與四張 NVIDIA s870 server 顯示卡串聯運算下比較，其加速最高可達 100 倍。但是在 Intel E8400 與 NVIDIA s870 server 單卡兩

者運算速度相較之下，僅有 12.6 倍加速效果。Fritzsche (2009) 藉由 GPU 求解二維方型穴流，將雷諾數固定為 1000 下，改變網格大小分別為 64×64、128×128 及 256×256，說明了網格大小於 64x64 下核心記算量之密度較小，故記憶體操作時間無法有效的被隱藏，在網格提高至 128x128 記憶體操作時間有效的被隱藏，其在網格 128x128 下有最高加速成效為 7.6 倍。

Komatitsch (2010) 透過高階有限元素法模擬地震波的傳播，並且透過單精度的 CUDA 與 MPI 所運算之結果進行準確性比較，透過不同 CPU 叢集架構下，得到的加速成果分別是 12 倍及 20 倍。

Kuznik (2009) 透過 GPU 求解 LBM (lattice Boltzmann methods)，並指出單倍的精度已足夠應付多數運算，且 GPU 是一種求解計算流體力學方面成本較低廉的高速運算系統。Klossa (2010) 透過 CUDA 求解波茲曼方程 (Boltzmann equation)，說明了儘管顯示卡的記憶體大小有一定的限制，但在大部分二維模式中慢速流或是非穩態模擬方面已然足夠，並對於網格大小及運算設備進行一連串比較。

1.3 研究目的

在諸多數值模式中網格點數對其模擬結果相當程度上的差異，透過網格加密可以得到更細部的結果，但模擬時間勢必會延長許多。現今諸多數值模式的發展已達到了三維架構，網格點數倍增，運算量也隨之增加，必

須仰賴多台高速電腦同時運算，2007 年 NVIDIA 提出 CUDA，CUDA 是以顯示卡內眾多運算核心做為平行化運算之技術，使得以較低成本換取高速運算效能。因此，藉由二維不可壓縮穴流的模擬來探討 GPU 加速運算效益，並且對 GPU 程式的撰寫部分，進行簡易的概述，以利推廣至淹水模式、三維動床及三維水理模式之開發。

1.4 研究方法

本研究先以 C 語言撰寫出求解二維穩態方形穴流之程式碼，隨後以 Ghia (1982) 研究之結果進行檢定驗證，隨後將程式碼逐步改寫成 CUDA，而後再進行一次檢定驗證，以確保在資料的平行分配計算上無誤。然後調整穴流之長寬比進行一連串模擬，以 Cheng (2006) 論文研究之結果進行檢定驗證，再進行程式碼的最終優化，隨後改變穴流長寬比例及網格大小，進行一連串運算效率之比較。

第二章 穴流理論基礎

本章介紹求解二維穩態穴流(cavity flow)之控制方程式，以及初始條件。

2.1 控制方程式

對於不可壓縮流體(incompressible flow)，其密度均勻且黏滯係數為常數。假設二維穴流初始時流場為靜止狀態，在上壁面施以一固定速度為 U 向右移動，其餘左、右及下壁面均為固定，如圖 2-1 所示。連續方程式及動量方程式分別為(2-1)式和(2-2)式所示：

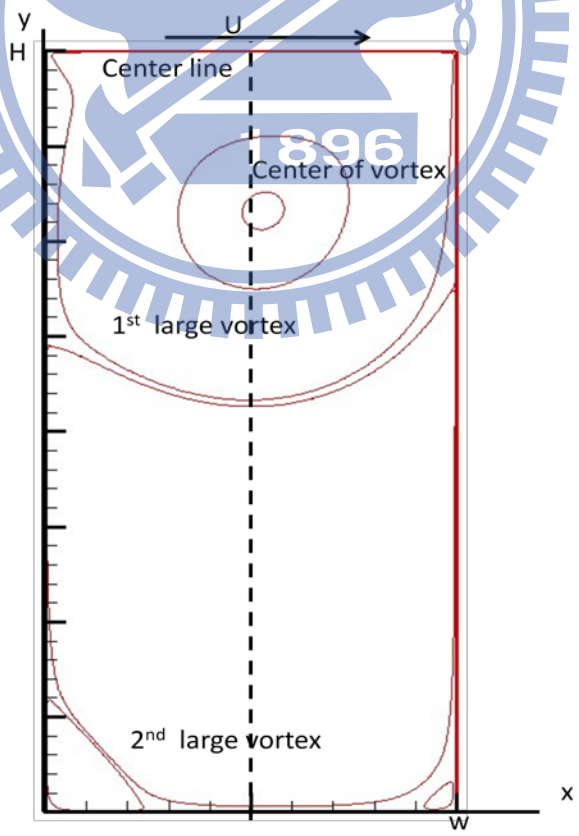


圖 2-1 穴流簡圖

連續方程式(continuity equation)

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{2-1}$$

動量方程式(momentum equation)

$$\frac{\partial \vec{U}}{\partial t}+\left(\vec{U}\cdot\nabla\right)\vec{U}=-\frac{\nabla p}{\rho}+\nu\nabla^2\vec{U} \tag{2-2}$$

x 方向之動量方程式

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+\frac{1}{\rho}\frac{\partial p}{\partial x}=\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{2-3}$$

y 方向之動量方程式

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+\frac{1}{\rho}\frac{\partial p}{\partial y}=\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{2-4}$$

式中$\vec{U}=(u,v)$分別為 x 與 y 方向的速度(velocity)、p 為流場壓力(pressure) 、ρ 為流體密度(density)、ν為流體黏滯係數(viscosity)。

將上述公式進行無因次化分析：

$x^*=\frac{x}{L}$ ， $y^*=\frac{y}{L}$ ， $t^*=\frac{tu_\infty}{L}$ ， $u^*=\frac{u}{u_\infty}$

$v^*=\frac{v}{u_\infty}$ ， $p^*=\frac{p}{\rho_\infty u_\infty^2}$ ， $Re=\frac{\rho_\infty u_\infty L}{\mu_\infty}=\frac{u_\infty L}{\nu}$

式中，上標*：無因次參數；u_∞：參考速度(上壁面抽動速度 U)；

L：特性長度(穴流寬度 W)。

利用上式無因次化參數，(2-1)、(2-3)、(2-4)式可化為：

連續方程式

$$\frac{\partial u^*}{\partial x^*}+\frac{\partial v^*}{\partial y^*}=0 \tag{2-5}$$

x 方向之動量方程式

$$\frac{\partial u^*}{\partial t^*} + u^* \frac{\partial u^*}{\partial x^*} + v^* \frac{\partial u^*}{\partial y^*} + \frac{1}{\rho^*} \frac{\partial p^*}{\partial x^*} = \frac{1}{Re} \left(\frac{\partial^2 u^*}{\partial x^{*2}} + \frac{\partial^2 u^*}{\partial y^{*2}} \right) \quad (2\text{-}6)$$

y 方向之動量方程式

$$\frac{\partial v^*}{\partial t^*} + u^* \frac{\partial v^*}{\partial x^*} + v^* \frac{\partial v^*}{\partial y^*} + \frac{1}{\rho^*} \frac{\partial p^*}{\partial y^*} = \frac{1}{Re} \left(\frac{\partial^2 v^*}{\partial x^{*2}} + \frac{\partial^2 v^*}{\partial y^{*2}} \right) \quad (2\text{-}7)$$

其邊界條件可以表示為：

當 x=0 之左壁面時：

$$u^*(0, y^*, t^*) = 0 \quad (2\text{-}8)$$

$$v^*(0, y^*, t^*) = 0 \quad (2\text{-}9)$$

當 x=W 之右壁面時：

$$u^*(W, y^*, t^*) = 0 \quad (2\text{-}10)$$

$$v^*(W, y^*, t^*) = 0 \quad (2\text{-}11)$$

當 y =0 之下壁面時：

$$u^*\left(x^*, 0, t^*\right) = 0 \quad (2\text{-}12)$$

$$v^*\left(x^*, 0, t^*\right) = 0 \quad (2\text{-}13)$$

當 y =H 之上壁面時：

$$u^*\left(x^*, H, t^*\right) = U \quad (2\text{-}14)$$

$$v^*\left(x^*, H, t^*\right) = 0 \quad (2\text{-}15)$$

起始條件可假定為：

$$u^*\left(x^*, y^*, t^*\right) = 0 \quad (2\text{-}16)$$

$$v^*\left(x^*, y^*, t^*\right) = 0 \quad (2\text{-}17)$$

倘若直接以原始變數 u、v 及 p 進行求解連續方程式及動量方程式，可能會

因為連續方程式中沒有出現壓力項變數，導致數值計算上的困難，因此將動量方程式及能量方程式改寫為渦度流線型態之方程式。

2.2 渦度流線型態之控制方程式

連續方程式及動量方程式可改寫為流線函數方程式與渦度傳輸方程式，推導如下所示：

流線函數 Ψ

$$u = \frac{\partial \psi}{\partial y} \tag{2-18}$$

$$v = -\frac{\partial \psi}{\partial x} \tag{2-19}$$

渦度 $\vec{\Omega}$

$$\vec{\Omega} = \nabla \times \vec{V} \tag{2-20}$$

$$\Omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{2-21}$$

將(2-18)、(2-19)式代入(2-21)，可得

$$\nabla^2 \Psi = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\Omega \tag{2-22}$$

上式中，Ψ 為流線函數，Ω 為渦度。

(2-6)式對 x 偏微分減去(2-7)對 y 偏微分，可得：

$$\frac{\partial \Omega}{\partial t} + u\frac{\partial \Omega}{\partial x} + v\frac{\partial \Omega}{\partial y} = \frac{1}{R_e}\left(\frac{\partial^2 \Omega}{\partial x^2} + \frac{\partial^2 \Omega}{\partial y^2}\right) \tag{2-23}$$

上式為渦度傳輸方程式。

定義無因次參數項如下：

$$x^* = \frac{x}{L} \quad , \quad y^* = \frac{y}{L} \quad , \quad t^* = \frac{tu_\infty}{L} \quad , \quad u^* = \frac{u}{u_\infty}$$

$$v^* = \frac{v}{u_\infty} \quad , \quad \Omega^* = \frac{\Omega L}{u_\infty} \quad , \quad \Psi^* = \frac{\Psi}{u_\infty L}$$

式中，上標*：無量綱參數；u_∞：參考速度(上壁面抽動速度 U)；

L：特性長度(穴流寬度 W)。

利用上述無因次參數，將(2-22)式無因次化後得到

$$\frac{\partial^2 \psi^*}{\partial x^{*2}} + \frac{\partial^2 \psi^*}{\partial y^{*2}} = -\Omega^* \tag{2-24}$$

同理將(2-23)是無因次化後可得

$$\frac{\partial \Omega^*}{\partial t^*} + u^* \frac{\partial \Omega^*}{\partial x^*} + v^* \frac{\partial \Omega^*}{\partial y^*} = \frac{1}{R_e}\left(\frac{\partial^2 \Omega^*}{\partial x^{*2}} + \frac{\partial^2 \Omega^*}{\partial y^{*2}}\right) \tag{2-25}$$

式中 Re 為雷諾數可表示為 $Re = \frac{u_\infty L}{\nu}$。

(2-25)式中，壓力項已經被消去，所以不用考慮壓力造成的影響。

2.3 邊界條件及起始條件

根據一開始的描述，設定除了方形槽上壁面以速度 U 向右移動之外，其餘壁面均保持靜止狀態，在滿足無滑動條件下之情況下，渦度流線型態的邊界條件可以假設為

當 x=0 之左壁面時：

$$\psi^*(0, y^*, t^*) = 0 \tag{2-26}$$

$$\Omega^*(0, y^*, t^*) = -\frac{\partial^2 \psi^*}{\partial x^{*2}}\bigg|_{x^*=0} \tag{2-27}$$

$$v^*(0,y^*,t^*) = -\frac{\partial\psi^*}{\partial x^*}\Big|_{x^*=0} = 0 \quad (2\text{-}28)$$

當 x=w 之右壁面時：

$$\psi^*(w,y^*,t^*) = 0 \quad (2\text{-}29)$$

$$\Omega^*(w,y^*,t^*) = -\frac{\partial^2\psi^*}{\partial x^{*2}}\Big|_{x^*=\frac{w}{W}} \quad (2\text{-}30)$$

$$v^*(w,y^*,t^*) = -\frac{\partial\psi^*}{\partial x^*}\Big|_{x^*=\frac{w}{W}} = 0 \quad (2\text{-}31)$$

當 y=0 之下壁面時：

$$\psi^*(x^*,0,t^*) = 0 \quad (2\text{-}32)$$

$$\Omega^*(x^*,0,t^*) = -\frac{\partial^2\psi^*}{\partial y^{*2}}\Big|_{y^*=0} \quad (2\text{-}33)$$

$$u^*(x^*,0,t^*) = \frac{\partial\psi^*}{\partial y^*}\Big|_{y^*=0} = 0 \quad (2\text{-}34)$$

當 y=H 之上壁面時：

$$\psi^*(x^*,H,t^*) = 0 \quad (2\text{-}35)$$

$$\Omega^*(x^*,H,t^*) = -\frac{\partial^2\psi^*}{\partial y^{*2}}\Big|_{y^*=\frac{H}{W}} \quad (2\text{-}36)$$

$$u^*(x^*,H,t^*) = \frac{\partial\psi^*}{\partial y^*}\Big|_{y^*=\frac{H}{W}} = 1 \quad (2\text{-}37)$$

渦度流線型態之起始條件可以假設為：

$$\psi^*(x^*,y^*,0) = 0 \quad (2\text{-}38)$$

$$\Omega^*(x^*,y^*,0) = 0 \quad (2\text{-}39)$$

第三章 數值方法

本章推導數值模式，並敘述其演算過程，藉以求得二維穩態穴流控制方程式的近似解。

3.1 有限差分法簡介與特點

自然界中許多現象均是以偏微分方程的型式來描述，若是能求得方程式的解，則可以對這些現象進行定性定量的分析。流體流動現象大量存在於自然界及多種工程領域之中，所有過程均滿足質量守恆、動量守恆及能量守恆等基本物理定律。計算流體力學（computational fluid dynamics）則是透過電腦數值計算與圖像顯示，對流體流動、熱傳導等物理現象所做的分析。在過去六十年來科學計算上有長足的進步，數值模擬、理論分析及實驗法這三種方法組成了研究流體流動問題的完整體系。數值模擬必須透過相關理論分析，再以模型問題進行初步驗證，最後在與實驗法所得的真實數據進行比較。

經過六十多年的發展，計算流體力學演變出多種數值解法。在這些方法之間的主要區別則是在於控制方程式的離散方式。根據離散的原理不同，計算流體力學大致上可分為三個分支：

- 有限差分法(finite difference method, FDM)
- 有限元素法(finite element method, FEM)

- 有限體積法(finite volume method, FVM)

有限差分法為應用最早，最經典的 CFD 方法，將其求解區域劃分為差分網格，運用有限的網格節點取代連續的求解區，再將偏微分方程的導數運用泰勒級數展開，推導出含有離散點上有限個未知數的差分方程組。在求出差分方程組的解，就是微分方程定解問題的數值近似解。是一種將微分方程變成代數問題的近似數值解法。藉由此基礎發展出來的方法有 PIC 法(particle-in-cell)、MAC 法(marker-and-cell)法，及美國佛羅里達大學陳景仁教授所提出的有限解析法(finite analytic method)等。

有限元素法則採用了有限差分法中離散處理的方式，並在變分法中選擇逼近函數對區域進行積分的方法。有限元素法求解速度相對於有限差分法及有限體積法慢，因此在應用方面並不是特別廣泛。在有限元素法的基礎架構上，英國的 Brebbia 等提出了邊界元素法、混合元素法等。

有限體積法則是將計算區域內劃分成為一系列控制體積，將欲求之微分方程對每一個控制體積積分得到其離散方程。有限體積法的求解關鍵是在於導出離散方程式的過程中，必須對界面上的欲求函數本身及其導數的分佈做出某種型式的假定。有限體積分所導出的離散方程式可以保證其方程式具有守恆性質，且離散方程式的物理意義較於明確，其計算量相對較小。

3.2 離散化 Navier-Stokes 方程組

本小節描述如何透過數值方法解二維 Navier-Stokes 方程組，透過先前公式推導可得知 Navier-Stokes 方程組離散成為穩態下渦度流線型態分別為(2-24)式及(2-25)式。

$$\frac{\partial^2\psi^*}{\partial x^{*2}}+\frac{\partial^2\psi^*}{\partial y^{*2}}=-\Omega^* \tag{2-24}$$

$$\frac{\partial\Omega^*}{\partial t^*}+u^*\frac{\partial\Omega^*}{\partial x^*}+v^*\frac{\partial\Omega^*}{\partial y^*}=\frac{1}{Re}\left(\frac{\partial^2\Omega^*}{\partial x^{*2}}+\frac{\partial^2\Omega^*}{\partial y^{*2}}\right) \tag{2-25}$$

3.2.1 時間項離散

在(2-25)式中之時間項，在此區域內可被均分為 N 各長度為Δt的區域，採用向前差分模式進行離散

$$\frac{\partial\Omega^*}{\partial t^*}=\frac{\Omega^{*n+1}-\Omega^{*n}}{\Delta t^*} \tag{3-1}$$

3.2.2 區域內各點離散

假定本研究之穴流模擬區域如圖 3-1 所示，可將其劃分為$[0,x_{end}]\times[0,y_{end}]$，在 x 方向上可被均分為$i_{max}$個長度為$\Delta x$的區域，y 方向上可被均分為$j_{max}$個長度為$\Delta y$的區域，關係式如下

$$x_i=i\Delta x\ ,\ i\in[0,i_{max}]\ ,\ y_j=j\Delta y\ ,\ j\in[0,j_{max}]$$

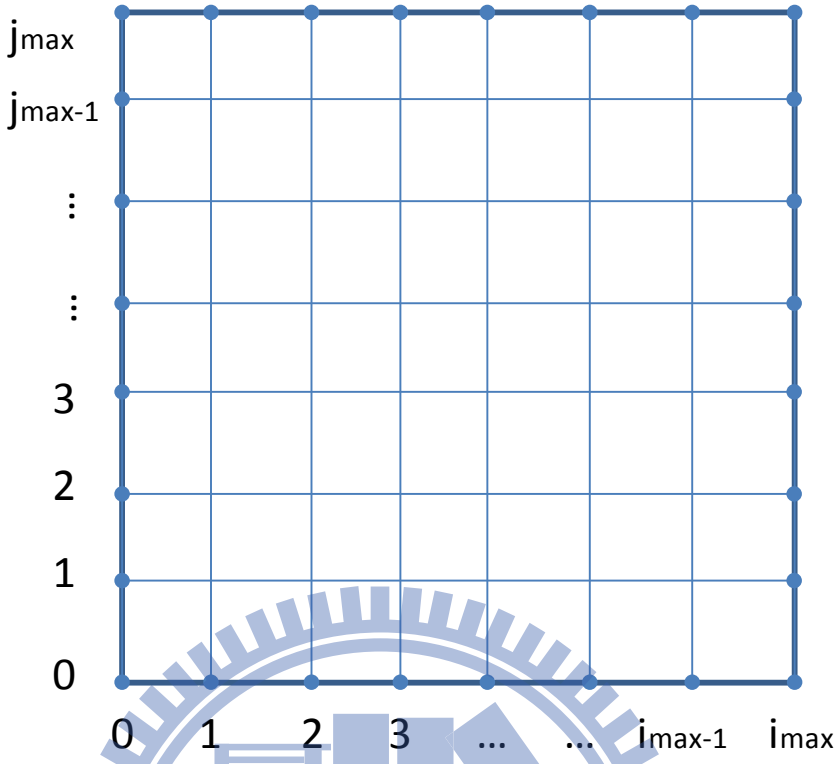


圖 3-1 區域離散示意圖

在各點離散方面均採用中央差分法進行離散，離散(2-24)式、(2-25)式可得：

流線控制方程式

$$\frac{\psi^{*}_{i+1,j}{}^{n}-2\psi^{*}_{i,j}{}^{n}+\psi^{*}_{i-1,j}{}^{n}}{(\Delta x)^2}+\frac{\psi^{*}_{i,j+1}{}^{n}-2\psi^{*}_{i,j}{}^{n}+\psi^{*}_{i,j-1}{}^{n}}{(\Delta y)^2}=-\Omega^{*}_{i,j}{}^{n} \tag{3-2}$$

渦度控制方程式

$$\frac{\Omega^{*}_{i,j}{}^{n+1}-\Omega^{*}_{i,j}{}^{n}}{\Delta t^{*}}+u^{*}{}_{i,j}{}^{n}\frac{\Omega^{*}_{i+1,j}{}^{n}-\Omega^{*}_{i-1,j}{}^{n}}{2\Delta x^{*}}+v^{*}{}_{i,j}{}^{n}\frac{\Omega^{*}_{i,j+1}{}^{n}-\Omega^{*}_{i,j-1}{}^{n}}{2\Delta y^{*}}$$
$$=\frac{1}{R_e}\left(\frac{\Omega^{*}_{i+1,j}{}^{n}-2\Omega^{*}_{i,j}{}^{n}+\Omega^{*}_{i-1,j}{}^{n}}{\Delta x^{*2}}+\frac{\Omega^{*}_{i,j+1}{}^{n}-2\Omega^{*}_{i,j}{}^{n}+\Omega^{*}_{i,j-1}{}^{n}}{\Delta y^{*2}}\right) \tag{3-3}$$

然而在渦度控制方程式中，存在有 u、v 此二速度項根據(2-18)、(2-19)式之定義將速度項表示為

$$u^{*}{}_{i,j}{}^{n}=\frac{\psi^{*}_{i,j+1}{}^{n}-\psi^{*}_{i,j-1}{}^{n}}{2\Delta y^{*}} \tag{3-4}$$

$$v^*_{i,j}{}^n = -\frac{\psi^*_{i+1,j}{}^n - \psi^*_{i-1,j}{}^n}{2\Delta x^*} \quad (3\text{-}5)$$

流線之邊界條件可表示為：

當 i=0 之左壁面時：

$$\psi^*(0,j) = 0 \quad (3\text{-}6)$$

當 $i=i_{max}$之右壁面時：

$$\psi^*(i_{max},j) = 0 \quad (3\text{-}7)$$

當 j=1 之下壁面時：

$$\psi^*(i,1) = 0 \quad (3\text{-}8)$$

當 $j=j_{max}$之上壁面時：

$$\psi^*(i,j_{max}) = 0 \quad (3\text{-}9)$$

而渦度之邊界條件可，可利用流線函數以及中央差分所得到，如下：

當 $j=j_{max}$之上壁面時：

運用中央差分法，可將(2-36)式及(2-37)式改寫成

$$\Omega^*(i,j_{max}) = -\frac{\psi^*(1,j)-2\psi^*(0,j)+\psi^*(-1,j)}{(\Delta y^*)^2} \quad (3\text{-}10)$$

$$\left.\frac{\partial\psi^*}{\partial y^*}\right|_{y^*=H} = \frac{2(\psi^*(1,j_{max})-\psi^*(-1,j_{max}))}{(\Delta y^*)^2} - 1 \quad (3\text{-}11)$$

將(3-10)式代入(3-11)式得到

$$\Omega^*(i,j_{max}) = \frac{2(\psi^*(i,j_{max})-\psi^*(i,j_{max}-1))}{(\Delta y^*)^2} - \frac{2}{\Delta y^*} \quad (3\text{-}12)$$

當 i=0 之左壁面時，同理，運用(2-27)與(2-28)式，可得

$$\Omega^*(1,j) = \frac{2(\psi^*(0,j)-\psi^*(1,j))}{(\Delta x^*)^2} \quad (3\text{-}13)$$

當 $i=i_{max}$之右壁面時，同理，運用(2-30)與(2-31)式，可得

$$\Omega^*(i_{max},j) = \frac{2(\psi^*(i_{max},j)-\psi^*(i_{max}-1,j))}{(\Delta x^*)^2} \quad (3\text{-}14)$$

當 j=1之下壁面時，同理，運用(2-33)與(2-34)式，可得

$$\Omega^*(i,1) = \frac{2(\psi^*(i,1)-\psi^*(i,2))}{(\Delta y^*)^2} \quad (3\text{-}15)$$

渦度與流線之起始條件如下:

$$\Omega^*(i,j) = 0 \quad (3\text{-}16)$$

$$\psi^*(i,j) = 0 \quad (3\text{-}17)$$

3.2.3 迭代法

迭代法的基本思想為將欲求解之線性方程組轉換為方便迭代的等價方程組，對其任選一組初始值$x_i^{(0)}$ $(i = 1,2\cdots n)$，按照其運算規則，不斷的對所得到的值進行修正，最終達到滿足精度要求的方程式近似解。

假設$A \in R^{i_{max}\times j_{max}}$為非奇異，$b \in R^{i_{max}}$，則在線性方成組 Ax=b 有唯一一解$x = A^{-1}b$，經過變換構造後，可得一等價同解方程組$x = Gx + d$，將上式改寫成為迭代式

$$x^{k+1} = Gx^k + d \quad (k = 1,2\cdots n) \quad (3\text{-}18)$$

選定初始向量$x_i^{(0)} = \left\{x_1^{(0)}, x_2^{(0)}, x_3^{(0)} \cdots x_{n-1}^{(0)}, x_n^{(0)}\right\}^T$，透過反覆迭代逐步逼近方程組的精確解，直到滿足精度為止。下述為本研究中所採用之迭代法演算過程。

演算法(1) Jacobi

for k=0 to kend
  for i= 1 to imax
    for j=1 to jmax

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j^{(K)}\right) \quad (3\text{-}19)$$

    end for
  end for
end for

演算法(2) Point Gauss-seidel

for k=0 to kend
  for i= 1 to imax
    for j=1 to jmax

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j>i} a_{ij}x_j^{(K)} - \sum_{j<i} a_{ij}x_j^{(K+1)}\right) \quad (3\text{-}20)$$

    end for
  end for
end for

演算法(3) Ponit Successive Over-Relaxation

for k=0 to kend
  for i= 1 to imax
    for j=1 to jmax

$$x_i^{(k+1)} = (1-w)x_i^{(k)} + \frac{w}{a_{ii}}\left(b_i - \sum_{j>i} a_{ij}x_j^{(K)} - \sum_{j<i} a_{ij}x_j^{(K+1)}\right) \quad (3\text{-}21)$$

    end for
  end for
end for

w 被定義為鬆弛係數

3.2.4 收斂條件

在將線性方程組轉換為方便迭代的等價方程組過程中，迭代公式(3-18)式將會收斂的充分必要條件是迭代矩陣 G 的譜半徑$\rho(G) < 1$ 。假若迭代矩陣 G 的一種範數$\|G\|<1$，則迭代式(3-25)將會收斂，且有誤差估計式

$$\left\|x^{\cdot} - x^{(k)}\right\| \leq \frac{\|G\|}{1-\|G\|}\left\|x^{(k)} - x^{(k-1)}\right\| \tag{3-22}$$

$$\left\|x^{\cdot} - x^{(k)}\right\| \leq \frac{\|G\|^{k}}{1-\|G\|}\left\|x^{(1)} - x^{(0)}\right\| \tag{3-23}$$

根據上述條件可知，當$\|G\|<1$ 時，其值越小，迭代收斂越快，在程式設計之中常用相鄰兩次迭代$\left\|x^{(k)} - x^{(k-1)}\right\| < \varepsilon$ (ε 為給定的精度要求)做為控制迭代結束的條件，本文所使用的收斂判斷如下：

$$\text{Error} = \sum_{\substack{i=1 \\ j=1}}^{\substack{i=i_{max} \\ j=j_{max}}} \left\|x^{(k+1)} - x^{(k)}\right\| \tag{3-24}$$

在 Error < ε 滿足收斂標準。

3.3 求解流程

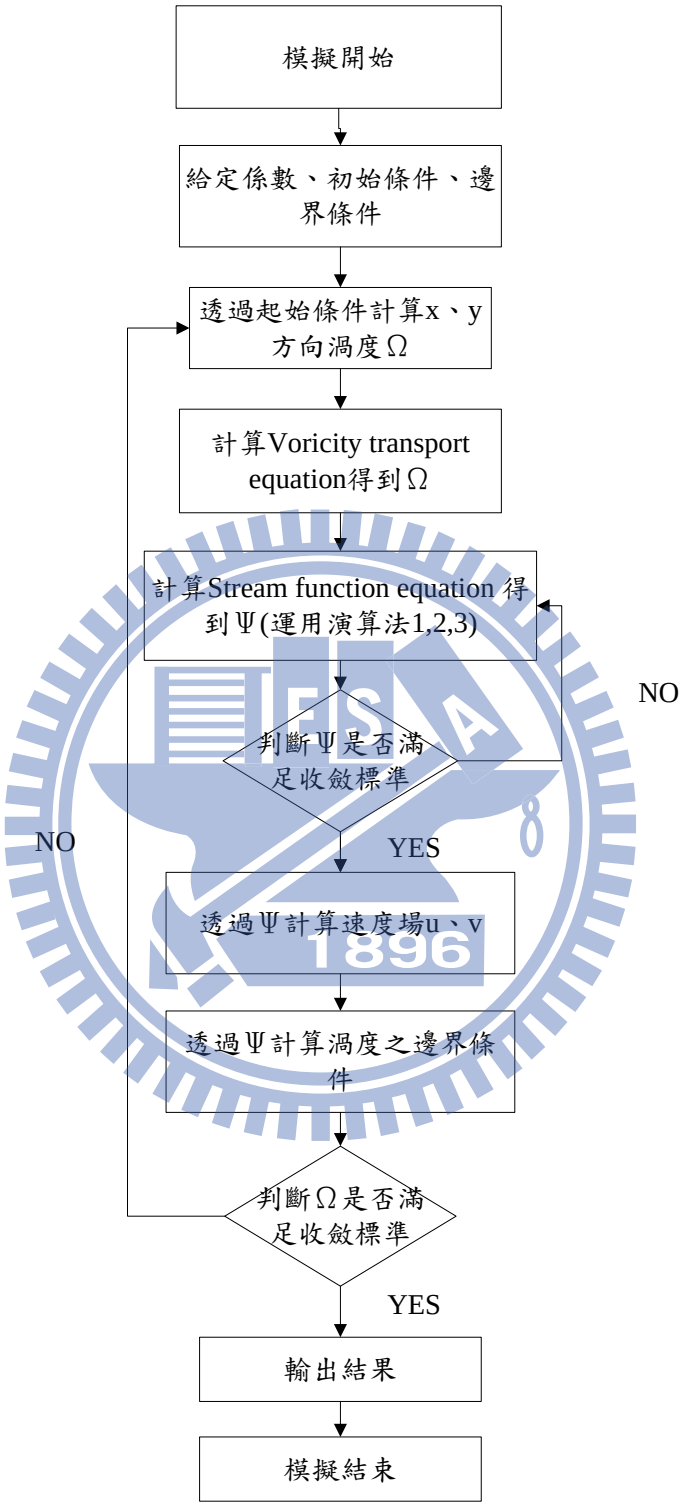


圖 3- 2 求解流程圖

第四章 GPU 理論

本章對 GPU 進行一系列概述，包含模型架構、簡易的編寫程式。

4.1 GPU 發展概要

目前，在個人電腦中的處理器可分為中央處理器(CPU)以及圖形處理器(GPU)。在傳統上 GPU 只負責影像處理部分，絕大部分的處理均交給 CPU。

二十一世紀所面臨的重要科技問題，像是基因工程、全球氣候準確預報、都市淹水的即時演算、核爆炸的模擬等，數據規模大小已經達到 TB 甚至於 PB 量級，因此對計算效率構成了嚴峻的挑戰。

GPU 在處理能力上和記憶體帶寬上相對於 CPU 而言有明顯優勢，且在成本以及功耗上也不需付出太多的代價，進而替這些問題提出了新的解決方法。由於影像處理具有極佳的平行性，使得 GPU 可以透過增加平行處理單元，跟記憶體控單元的方式，有效的提高處理能力以及記憶體頻寬。GPU 的設計者將夠多的晶體管當成執行單元，而不是像 CPU 一樣運用複雜的控制單元和緩存進而增加執行單元的執行效率。圖 4-1 為 CPU 與 GPU 中晶體管數量以及用途架構。

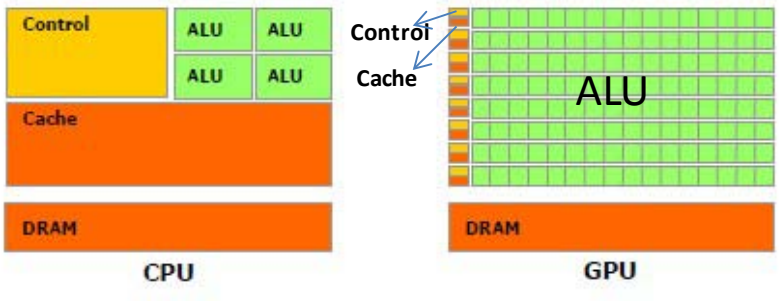


圖 4- 1 CPU 與 GPU 中晶體管數量以及用途架構

平行是一個廣義的概念，根據實行的方法不同可分為四種方式如表 4-1 所示：

表 4- 1 平行分層圖

單核指令集平行(ILP)	使單一個處理器執行單元可以同時執行多條指令
多核平行(multi-core parallel)	在一個晶片上擁有多個處理器核心，實現線程級平行(TLP)
多處理器平行(multi-processor parallel)	在一塊主機板上安裝多個處理器，實現線程和進程級平行
集群或分佈式平行(cluster/distributed parallel)	藉由網路實現大規模的集群或是分布式平行，每一個節點便是一台個人電腦

目前在個人電腦中的 CPU 為 2~8 個核心的多核處理器，在 GPU 方面則是擁有數十或上百個核心的眾核平行處理器，因此透過適當的程式撰寫，改

良演算法至使其可進行平行化運算，能有效的加強數值運算方面效率。

4.2 GPU 模型架構

在 2007 年 6 月，NVIDIA 推出了 CUDA(computer unified device architecture)。而 CUDA 則是一種將 GPU 作為數據平行計算設備的軟硬體連接架構。

4.2.1 主機與設備

在 CUDA 的模型之中，將 CPU 當成主機端(host 端)，並將 GPU 作為協助主機端處理運算的元件稱之為設備端(device 端)。然而因兩種處理器特性上差異，將 CPU 用處理邏輯性較強的事件處理以及串行運算，GPU 則執行高度線程化的平行處理。一旦在確定程式架構上何處可以進行平行化演算，便可以將這部分的計算工作交給 GPU 進行，然而在 GPU 上的 CUDA 平行運算的函數稱為 kernel，即在顯示卡上運行的程式碼區段。如圖 4-2 所示，一個完整的 CUDA 程式架構，是透過一系列的設備端 kernel 函數的平行處理，及主機端的串行處理步驟所組成。在圖 4-2 中可以看出，一個 kernel 函數中存在有兩個層面的平行，即 Grid 中的 Block 間平行，以及 Block 中的 Thread 平行，而兩層平行架構也是 CUDA 主要的賣點之一。

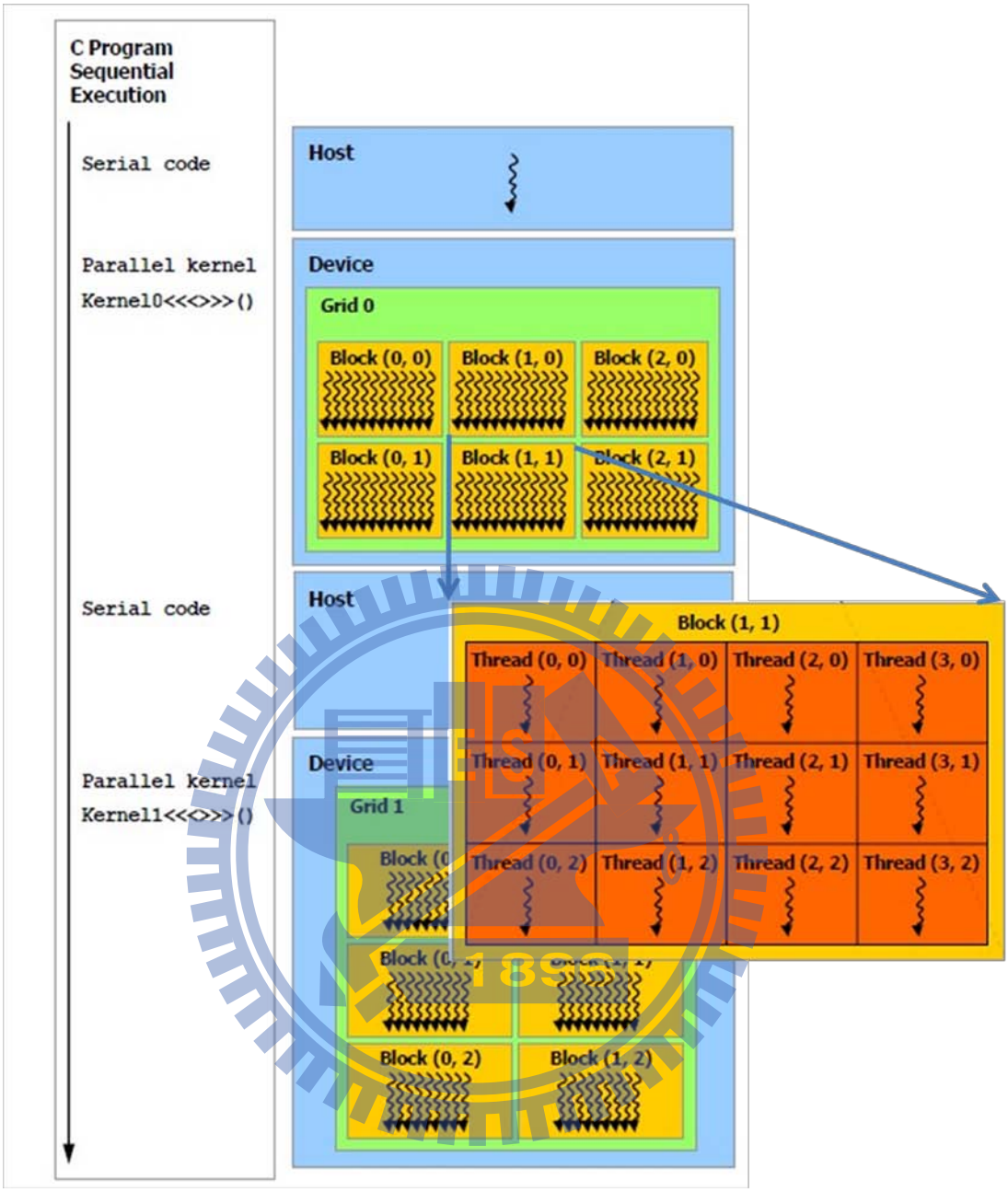


圖 4- 2 CUDA 架構模型

4.2.2 線程結構

CUDA 的平行化模型是將核心(kernel)交由一組格網(grid)執行，再將網格切成數個區塊(block)，然後每個區塊再分成數個執行緒(thread)，依次分發進行平行運算，如圖 4-3 所示。

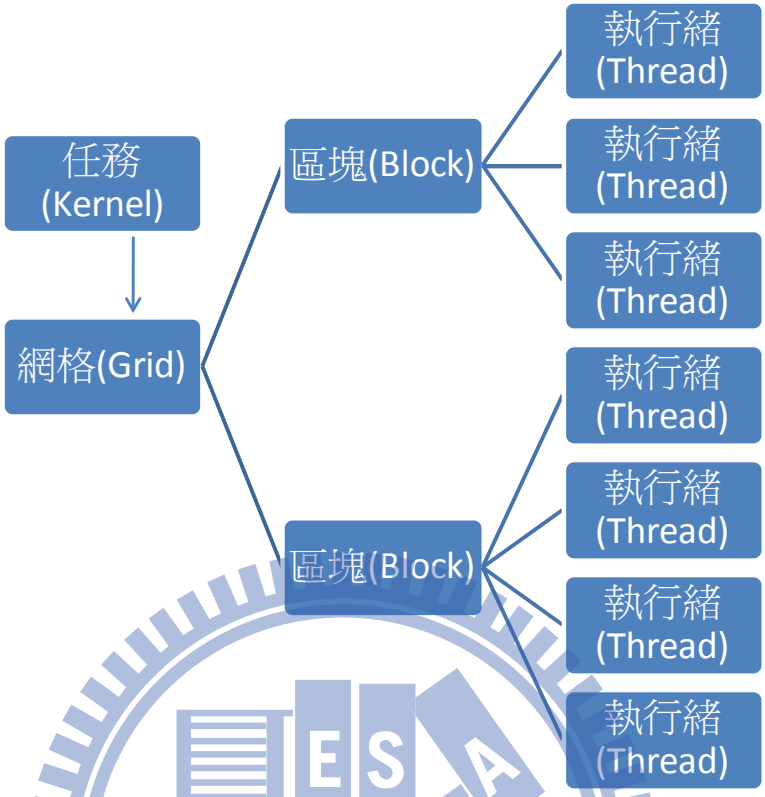


圖 4- 3 線程結構圖

在一個 Grid 中的 Block 數量僅與問題的規模有關，與實際上的硬體設備無關，高端的設備能夠擁有較多的 GPU 處理核心，致使其有更高的平行運算效能，通常會將 Block 的數量設置為至少是處理核心的數倍，這樣子才能夠有效的發揮 GPU 的運算能力，然而在設置 Block 大小中，要注意的是線程數量不能過超過 512 個。

4.2.3 硬體架構

GPU 是以多處理器(SM , stream multiprocessor)為群組，每個多處理器有 8 個單精度浮點數的串流處理器(SP , stream processor) 負責運算這些串流處理器，就是平常說的核心。如圖 4-4 所示。

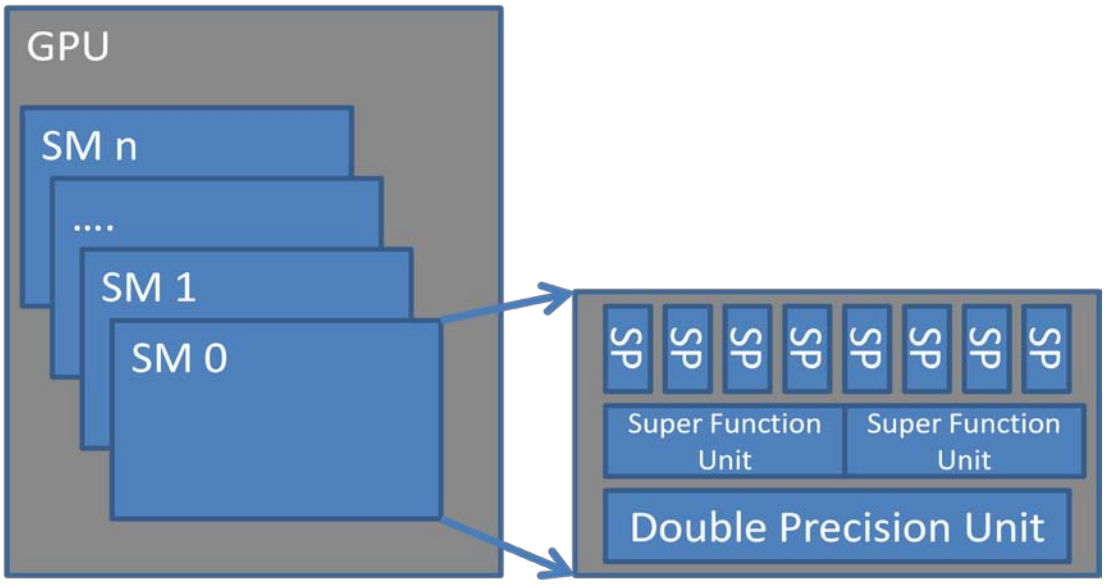


圖 4-4 GPU 計算單元概略圖

在 CUDA 之中的 kernel 函數實際上是以 Block 為單位執行的，在同一個 Block 之中數據是共享的，因此他們必須被配置到同一個 SM 之中，而 Block 中的每一個 Thread 則在被配置到一個 SP 上進行運作。

對於 GPU 記憶體頻寬相對於 CPU 大的許多，則是由於顯示卡中記憶體顆粒部分直接焊在顯示卡的 PCB 板上，故信號傳遞的完整性較高並且可擁有較高之工作頻率。而在 GPU 之記憶體架構中可以分為六大主要記憶體，如圖 4-5 所示。在程式撰寫完畢後，可以透過不同的記憶體類型來針對程式進行優化，藉此提高程式計算效率，本研究主要使用的記憶體為 Global Memory 部分。

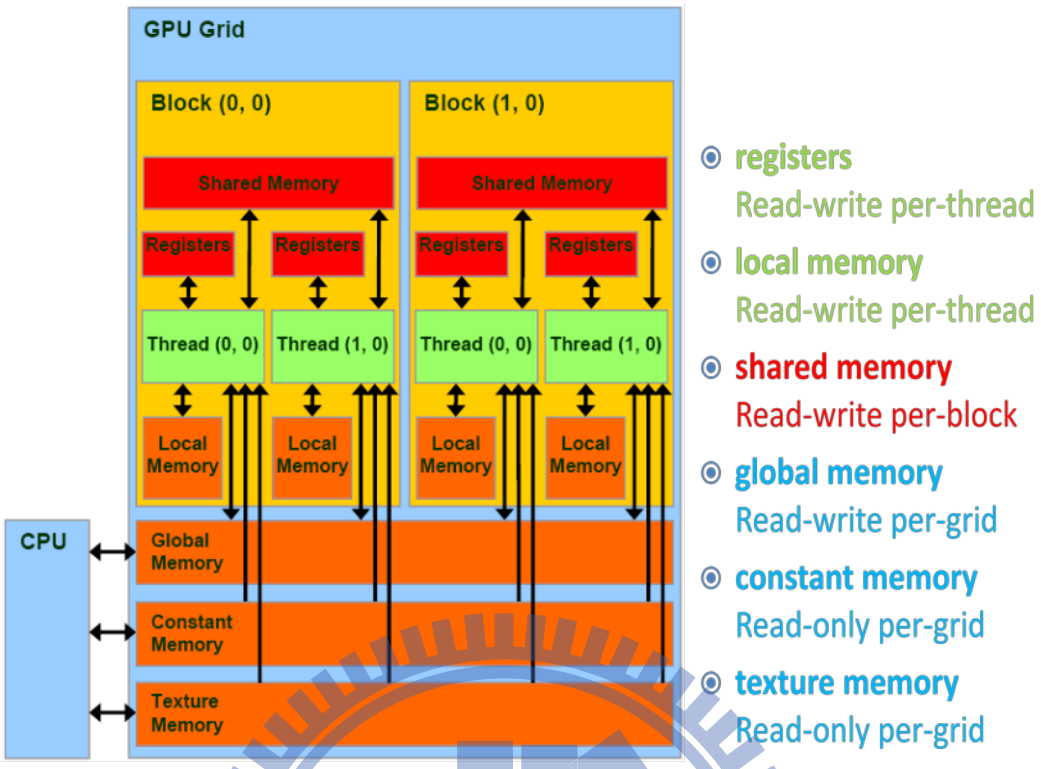


圖 4-5 GPU 內記憶體架構圖

4.3 簡易程式撰寫及效率評估

本節以簡易的向量加法為例子，使用隨機向量測試運算效能，選取三個計算模式進行一系列解說。

4.3.1 單一區塊，單一執行緒

可用來測試單一串流處理器 (SP，stream processor) 之計算效能，因僅使用單一個串流處理器，其程式碼寫法與 C 語言寫法一致。程式的運作情況如圖 4-6 示意。

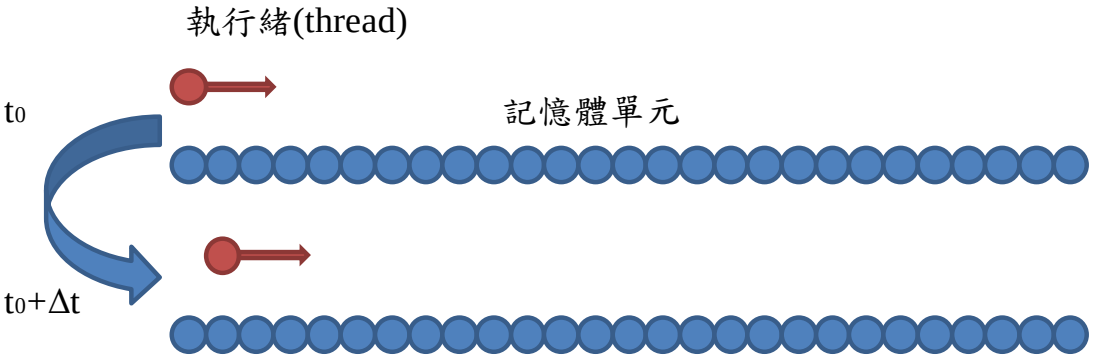


圖 4- 6 單一區塊，單執行緒

程式碼編寫部分

Block=1　代表了一個執行緒　Grid= 1　代表一個區塊

CPU

```
void add(float *a, float* b, float* c, int n)
{
      for (int k = 0; k<n; k++)
            c[k] = a[k] + b[k];
}


void main()
{
      .....
      add(a, b, c, n);
}
```

GPGPU

```
__global__ void Gadd(float *a, float* b, float* c, int n)
{
      for (int k = 0; k<n; k++)
            c[k] = a[k] + b[k];
}

void main()
{
      .....
      int Block = 1;
      int Grid =  1;
      Gadd <<< Grid, Block >>> (a, b, c, n);
}
```

4.3.2 單一區塊，多執行緒

可用來測試單一多串流處理器 (MP， multiprocessor) 之計算效能，不論包含多少執行緒，皆只在一個 MP 裡面進行運算，意即 MP 裡頭的八個 SP 輪流在這些執行緒中執行，其程式碼寫法將用到 threadIdx.x(區塊中的執行緒索引)、blockDim.x(區塊中的執行緒數量)。假設有八個執行緒，在同一

時刻所進行的運算為八個，到了下個時刻則一次前進八個單位長，如此運做下去。程式的運作情況如圖 4-7 示意。

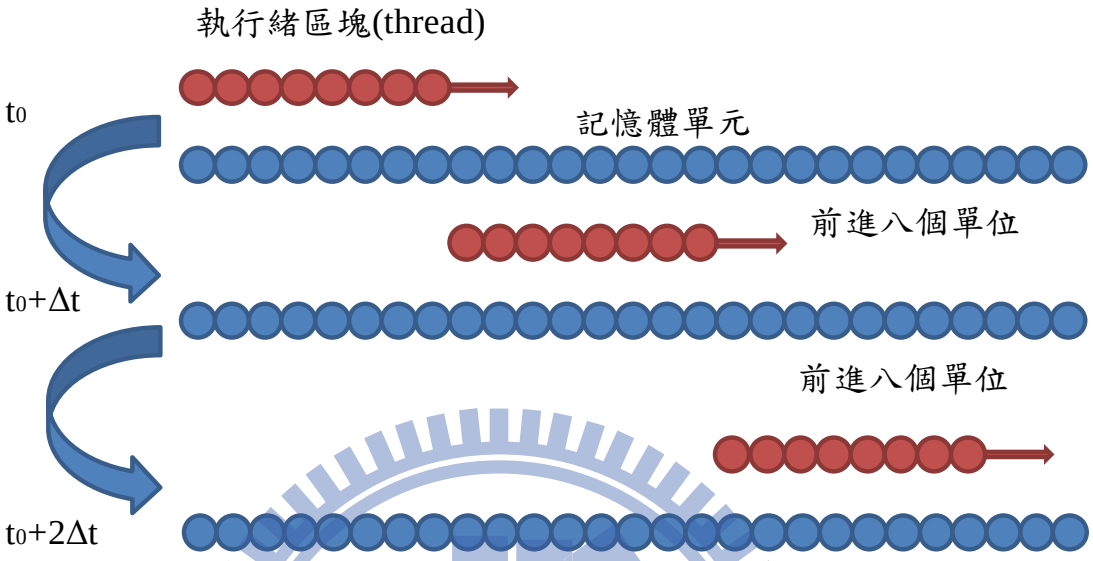


圖 4-7 單一區塊，多執行緒

程式碼編寫部分

Block=512　代表了 512 個執行緒　Grid= 1　代表 1 個區塊

CPU

```
void add(float *a, float* b, float* c, int n)
{
    for (int k = 0; k<n; k++)
        c[k] = a[k] + b[k];
}


void main()
{
    .....
    add(a, b, c, n);
}
```

GPGPU

```
__global__ void Gadd(float *a, float* b, float* c, int n)
{
    for( int k = threadIdx.x; k<n; k+=blockDim.x)
        c[k] = a[k] + b[k];
}


void main()
{
    .....
    int Grid = 1;
    int Block = 512;
    Gadd <<< Grid, Block >>> (a, b, c, n);
}
```

4.3.3 多區塊，多執行緒

當迴圈的數目小於 gridDim 與 blockDim 之乘積，前提為資料前後無相依性，便可以運用網格及區塊設定代替迴圈的部分，每一個執行緒只計算一次，即將整個 kernel 函數當成一平行化過的大迴圈。 (註:每個格網最大的區塊數量 gridDim 為 65535 個，每個區塊最大的執行緒數量為 512 個)，程式的運作情況如圖 4-8 示意。

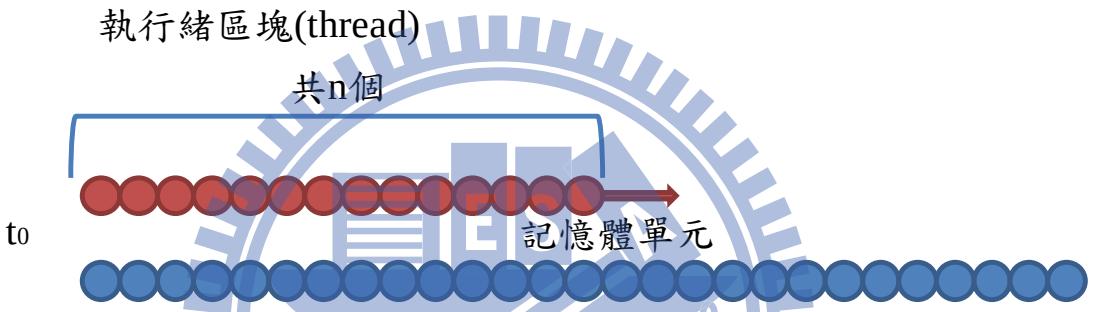


圖 4- 8 多區塊，多執行緒

程式碼編寫部分

Block=512　代表了 512 個執行緒

Grid=n/Block+1　代表 n/Block+1 個區塊

CPU

```
void add(float *a, float* b, float* c, int n)
{
    for (int k = 0; k<n; k++)
        c[k] = a[k] + b[k];
}

void main()
{
    .....
    add(a, b, c, n);
}
```

GPGPU

```
__global__ void Gadd(float *a, float* b, float* c, int n)
{
    int k = blockIdx.x * blockDim.x + threadIdx.x;

        c[k] = a[k] + b[k];
}

void main()
{
    .....
    int Block = 512;
    int Grid = n / Block + 1;
    Gadd <<< Grid, Block >>> (a, b, c, n);
}
```

4.3.4 效率評估

透過簡易的向量加法 C = A + B，來進行效能評估，假若向量 A 與向量 B 分別有1024^2個，將其運算數目設定為一百次進行效能評估。本次模擬設備條件如下：

中央處理器(CPU)	Intel® Core™2 Duo Processor E7400 (3M Cache, 2.80 GHz, 1066 MHz FSB)
記憶體(RAM)	DDR2-800　4G
顯示卡(GPU)	GTS250　256 MB　GDDR3 處理核心數共 128 個　單一時脈 1.8GHz
系統(OS)	Windows Xp　Sp3
編譯器(Copmiler)	Visual Studio 2005 掛載 CUDA 版本 3.0
編譯數值格式精度採用單倍精度	

(1) 單一區塊，單一執行緒

主機端運算時間 : 0.5 秒

設備端運算時間 : 36.89 秒

(2) 單一區塊，多執行緒

主機端運算時間 : 0.5 秒

設備端運算時間 : 0.078 秒

(3) 多區塊，多執行緒

主機端運算時間 : 0.5 秒

設備端運算時間 : 0.0016 秒

透上述模擬可以發現，在單執行緒單區塊中，即 1 個 SP 單元進行運算，GPU 的運算效率可以說是相當低落，CPU 整體運算效能為 GPU 的 7.4 倍左右。隨後以單一區塊多執行緒，即 1 個 MP 單元進行運算，GPU 的運算效率已有初步成效大約為 CPU 的 6.4 倍，隨後透過多區塊，多執行緒進行運算，GPU 運算效能為 CPU 之 31.3 倍左右，已是相當可觀。

4.3 透過 GPU 解 Navier-Stokes 相關演算法

在本研究中將流線方程式的解法稍做改寫，以及針對數值四則運算過程中採用了簡約算法，達到平行化運算的概念。

在運用 GPU 進行計算時，在主機端與設備端兩端進行資料交換時所花費之時間，與計算數據量的多寡有關，為了避免花費多餘的資料傳輸時間，

盡量將大量的數據運算在 GPU 上進行，並且盡可能的將設備端傳輸至主機端的資料量少量化。因此在最後以式(3-24)判定是否滿足收斂條件時，將運算數據由設備端拷貝至主機端來進行運算，故在此透過簡約算法(reduction algorithm) 將數值在設備端上先行運算，以減少傳輸至主機端的數據量。

4.4.1 流線方程式

此方程式以 Jacobi 法進行離散，在計算次數上，一次便運行兩次，以減少資料的交換，再運行兩次計算後，便透過式(3-24)判斷是否滿足收斂條件。

程式碼如下：

```
__global__ void Laplace_d(float *A, float *B, float *vo1, float dx, float dy, float bata)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x ;
  int j = blockIdx.y * blockDim.y + threadIdx.y ;
  int index,index1,index2,index3,index4;

  index = i*n + j;     index1= i*n + j + 1;     index2= i*n + j - 1;
  index3= (i+1)*n + j;    index4= (i-1)*n + j;
  if(i>0 && i<m-1 && j>0 && j<n-1) {

  B[index] = (1/(2*(1+bata*bata)))*( (dx*dx)*vo1[i*n+j]+
                        (bata*bata)*(A[index1] + A[index2]) + A[index3] + A[index4] );

  }
}
void main(){

 bata=dx/dy;
 int ThreadsPerBlock=8,iterations=2,iter;
    dim3 dimBlock( ThreadsPerBlock, ThreadsPerBlock );
    dim3 dimGrid( ceil(float(m)/float(dimBlock.x)), ceil(float(n)/float(dimBlock.y)) );
...
while(iter<iterations)
  {
   Laplace_d<<<dimGrid, dimBlock>>>(gps1,gps2,gvo1,dx,dy,bata);
   Laplace_d<<<dimGrid, dimBlock>>>(gps2,gps1,gvo1,dx,dy,bata);
   iter+=2;
  }
...
}
```

4.4.2 簡約算法(reduction algorithm)

傳統的樹狀簡約算法如圖 4-9 所示，假若有八個元素做累加動作，則先兩兩相加，重複動作到累加完畢為止。

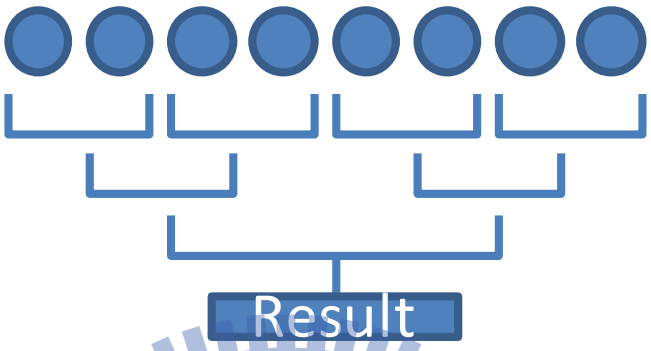


圖 4- 9 樹狀計算過程

GPU 的簡約計算類似傳統樹狀簡約計算，但是其運算過程有些改變，運算過程如圖 4-10。

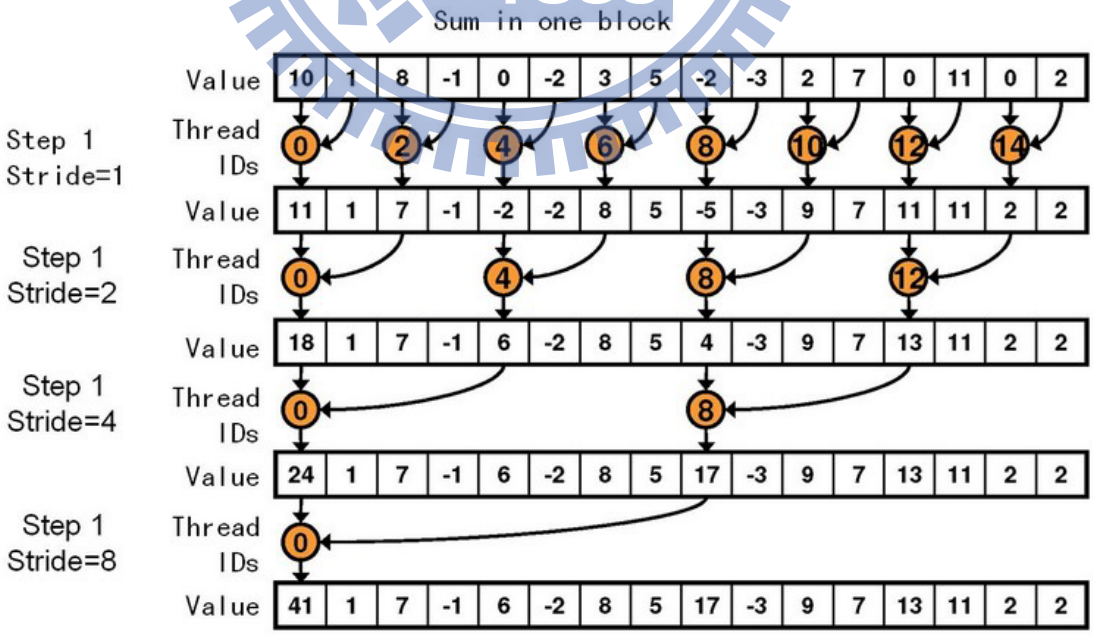


圖 4- 10 GPU 簡約計算過程

解說如下：

- 在第一次循環(Stride=1)時，僅有 Thread ID=0、2、4…14 線程的運算，即程式碼中 idate[i]與其往後 1 個單位的元素進行加總動作。
- 在第二次循環(Stride=2)時，僅有 Thread ID=0、4、8…線程的運算，即程式碼中 idate[i]與其往後 2 個單位的元素進行加總動作。
- 以此類推。
- 直到最後一次時假若為(Stride=8)，僅有 Thread ID=0 此線程進行運算，即程式碼中 idate[i]與其往後 8 個單位的元素進行加總動作，其和即為輸入之 28 組數據之和。

程式碼如下：

```
__global__ void reduce(double* idata, double* result)
{
   int tid = threadIdx.x;
   int i = blockDim.x*blockIdx.x + threadIdx.x;

   for(unsigned int s = 1; s < blockDim.x; s *= 2)
   {
      if (tid % (2*s) == 0)
      {
         idata[i] += idata[i + s];
      }
      __syncthreads();
   }
   if (tid == 0) result[blockIdx.x] = idata[i];
}
void main(){
#define BLOCKSIZE 128
 bata=dx/dy;
 int ThreadsPerBlock=8,iterations=2,iter;
     dim3 dimBlock( ThreadsPerBlock, ThreadsPerBlock );
     dim3 dimGrid( ceil(float(m)/float(dimBlock.x)), ceil(float(n)/float(dimBlock.y)) );
    int threadnum = BLOCKSIZE;   int blocknum = m*n/threadnum;
    int threadnum2 =  BLOCKSIZE;  int blocknum2 = blocknum/BLOCKSIZE;
...
                  GPUerrorB<<<dimGrid,dimBlock>>>(gps1,gps2,gerror);//(X(k+1)-X(k))
                  reduce<<<blocknum, threadnum>>>(gerror, dResult);
                  reduce<<<blocknum2, threadnum2>>>(dResult, dResult);

cudaMemcpy(hResult, dResult, sizeof(double)*blocknum2, cudaMemcpyDeviceToHost);
  for (i = 0; i < blocknum2; i++)
   {
            error_psor[0] += hResult[i];
   }...
}
```

第五章 模擬驗證與結果討論

本章首先以 Ghia et al. (1982) 穴流模擬結果進行模擬驗證，再調整穴流模擬範圍之長寬比，以 Cheng (2006) 模擬結果進行驗證，隨後探討 GPU 加速之成效。

5.1 模擬驗證與分析

5.1.1 模擬設備

中央處理器(CPU)	Intel® Core™2 Duo Processor E7400 (3M Cache, 2.80 GHz, 1066 MHz FSB)
記憶體(RAM)	DDR2-800　4G
顯示卡(GPU)	GTX 480　1536MB GDDR5 處理核心數共 480 個　單一時脈 1.4GHz
系統(OS)	Windows XP Service pack 3
編譯器(Compiler)	Visual Studio 2005 掛載 CUDA 版本 3.0
編譯數值格式精度採用單倍精度	

本研究採用單倍精度進行模擬，Kuznik (2009) 指出運用單倍精度已足夠應付目前多數的數值演算。而選用單精度做為數值儲存格式，因其在 GPU 目前所開發的編譯器架構上較為成熟並且有較高的運算效能，雙精度數值格式下 GPU 上運算成效較為低落，並且在有限的記體容量限制之下，採用單倍精度能夠存取較多數值。根據 IEEE-754 浮點數之定義可得知單倍精度下，有效數字可達 7 位，即精確度到小數點後第 6 位，在小數點後第 6 位

會產生數值的震盪，實則為電腦儲存格式之問題。因此在流線模擬成果繪製上僅輸出到10^{-7}。

5.1.2 方型穴流之模擬

拉蓋方穴流(lip-driven cavity flow)的幾何形狀如圖 5-1 所示，為一長寬比為一的矩型，當流體以均勻速度 U 由上壁面的左方流至右方，將在方穴裡面形成渦流，其餘三壁面邊界速度均為零，而透過不同的速度大小可以得到不同雷諾數下的渦度及流線模擬情況。

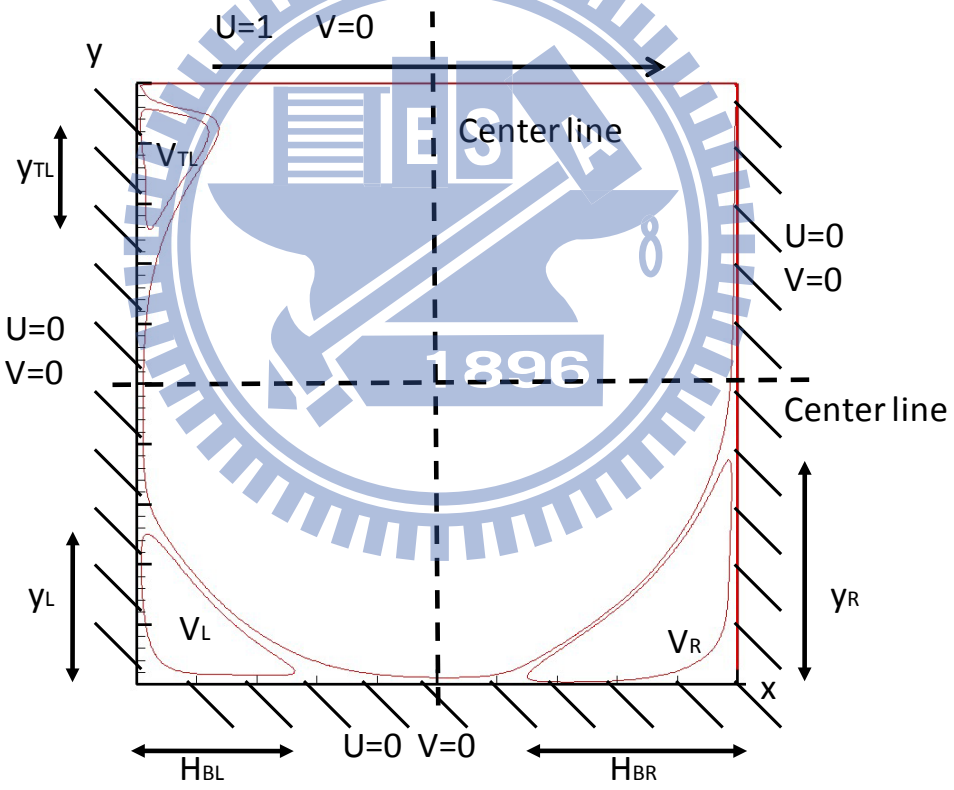


圖 5- 1 方穴流示意圖

根據 Ghia et al. (1982) 的研究，本研究分別於雷諾數 100、400、1000、3200、5000、7500 及 10000 進行模擬，採用 Ghia et al. (1982)模擬之網格數數大小，分別在雷諾數 100、400、1000 及 3200 部分採用網格數為 129×129，

雷諾數 5000、7500 及 10000 部分網格數採用 257×257。模擬至穩定狀態(steady state)，其流線與渦度模擬結果分別如圖 5-2 至圖 5-15 所示。

根據模擬結果可以發現到當雷諾數改變時將使流場結構產生變化，雷諾數越高則流場形狀越複雜，所形成的渦漩數量也會變多。在雷諾數 1000 以下時僅在下壁面左右兩側各產生一渦漩，隨後在雷諾數達到 3200，在左壁面上方又在產生一渦漩，而後雷諾數提高到 5000 之際，左上角之渦漩也更往中心點移動，而在中央渦漩部分隨著雷諾數增加而逐漸往幾何中心移動(x=1/2，y=1/2)。本研究模擬之流線與渦度趨勢圖與 Ghia 所模擬之成果有一樣的趨勢，而渦漩中心點也大置上位於相同的位置。

隨後根據 Ghia et al. (1982)之水平與垂直中心線上速度計算結果，做為比較之基準如圖 5-16 至圖 5-29 所示，由圖中可以發現雷諾數增加下，速度分布曲度變化隨之更加明顯。本文研究之模擬結果，與 Ghia 模擬結果之 u、v 速度相差無幾，而在將程式碼改編成 CUDA 演算之模式下，所求得之模擬成果亦相當準確。

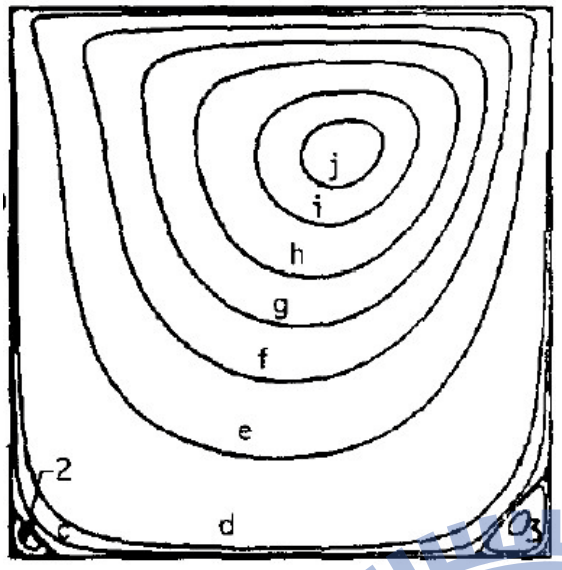
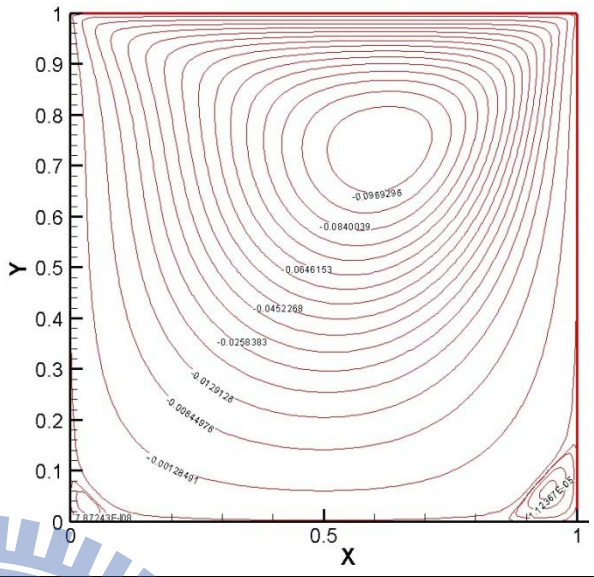
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 2 雷諾數 100 之流線比較圖

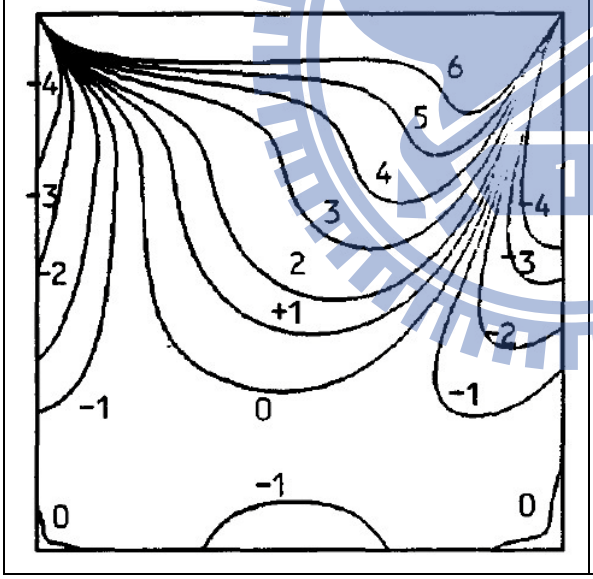
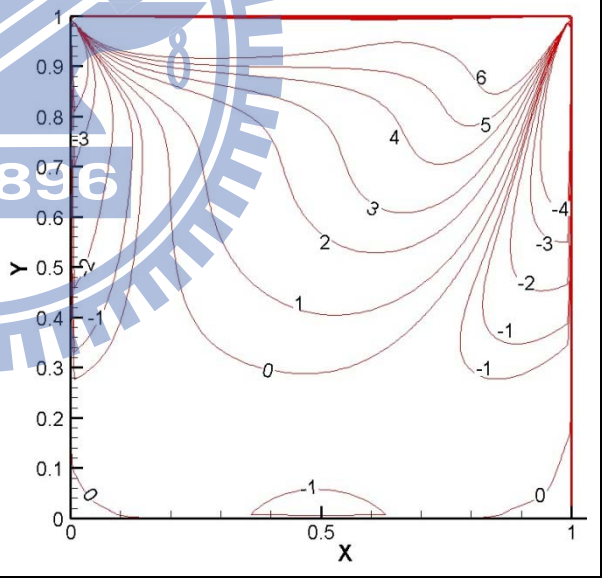
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 3 雷諾數 100 之渦度比較圖

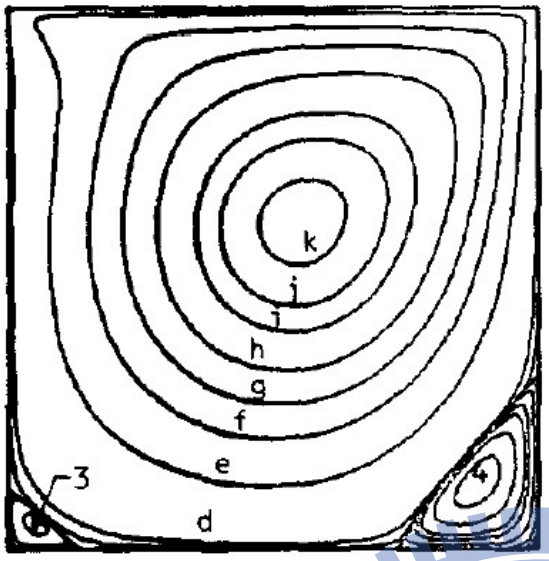
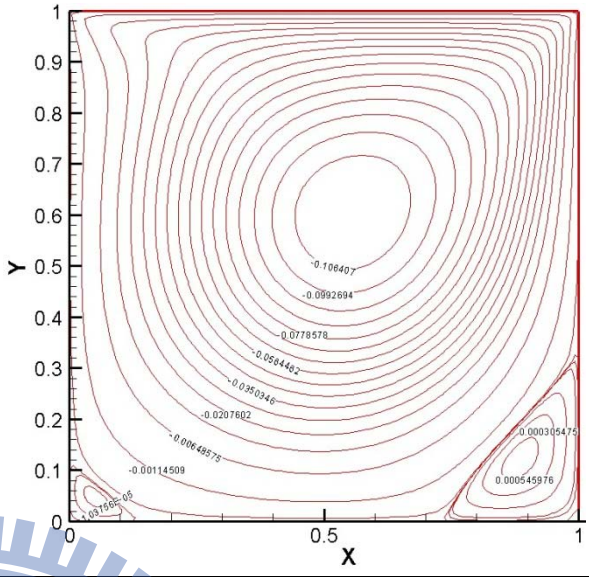
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 4 雷諾數 400 之流線比較圖

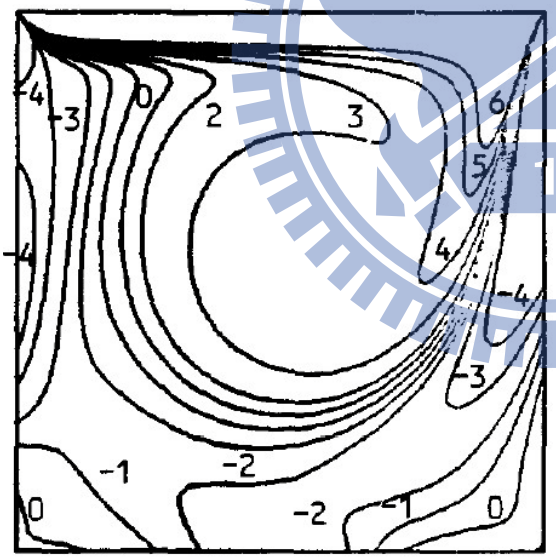
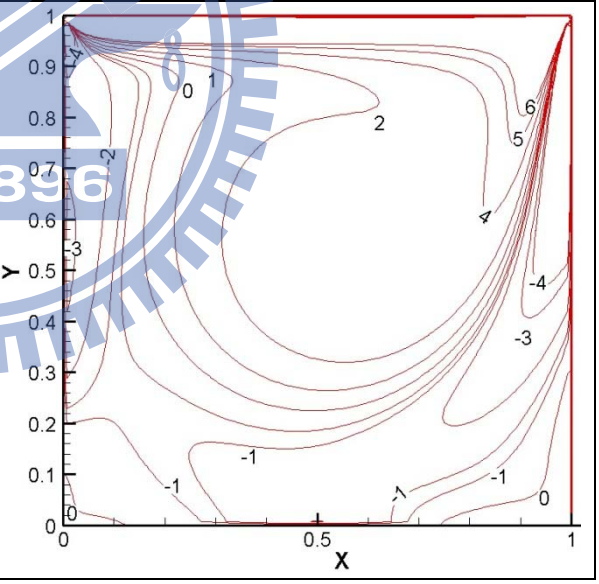
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 5 雷諾數 400 之渦度比較圖

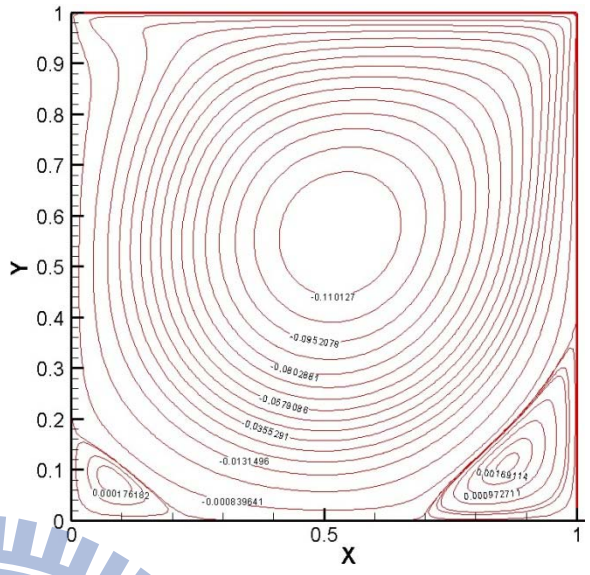
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 6 雷諾數 1000 之流線比較圖

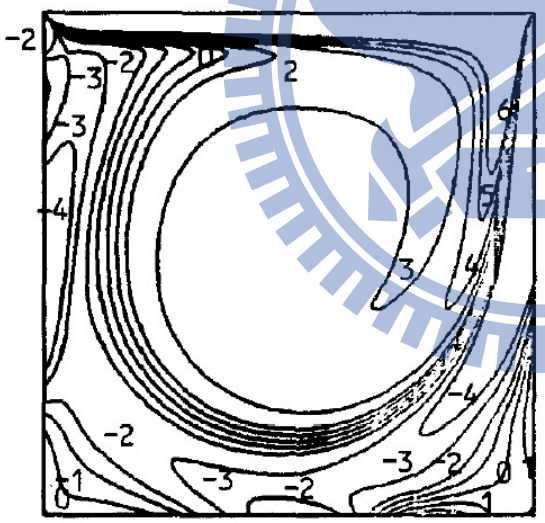
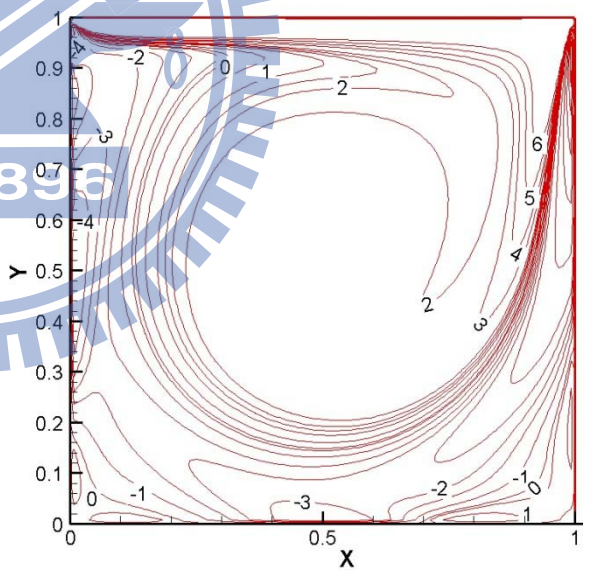
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 7 雷諾數 1000 之渦度比較圖

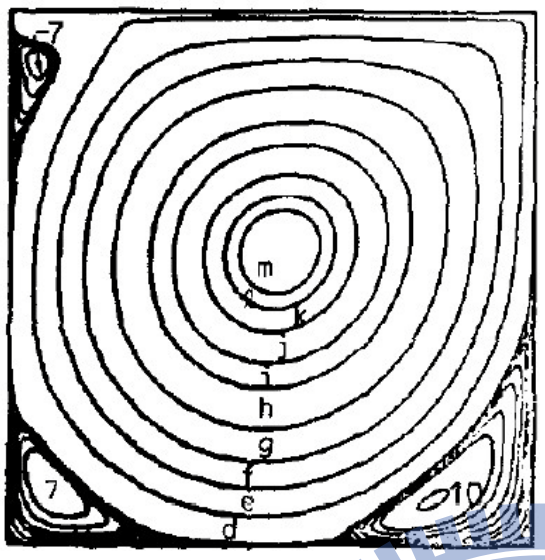
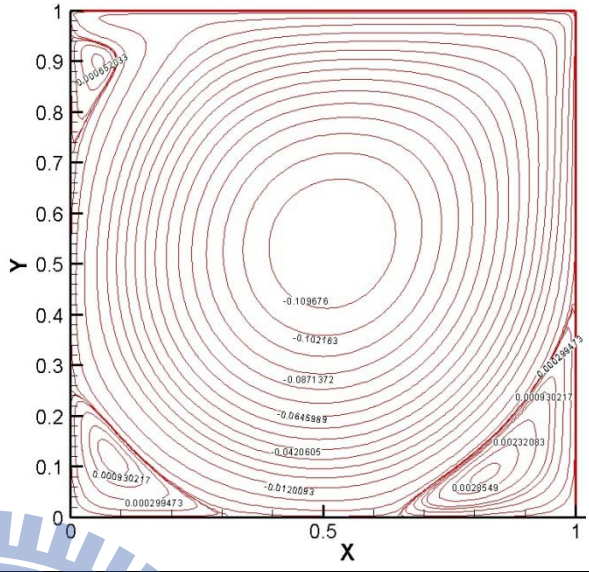
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 8 雷諾數 3200 之流線比較圖

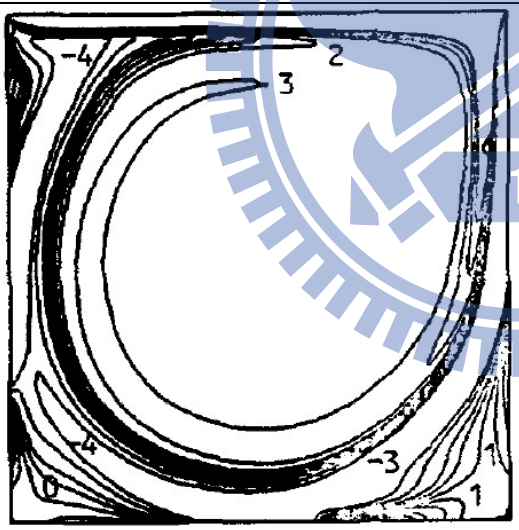
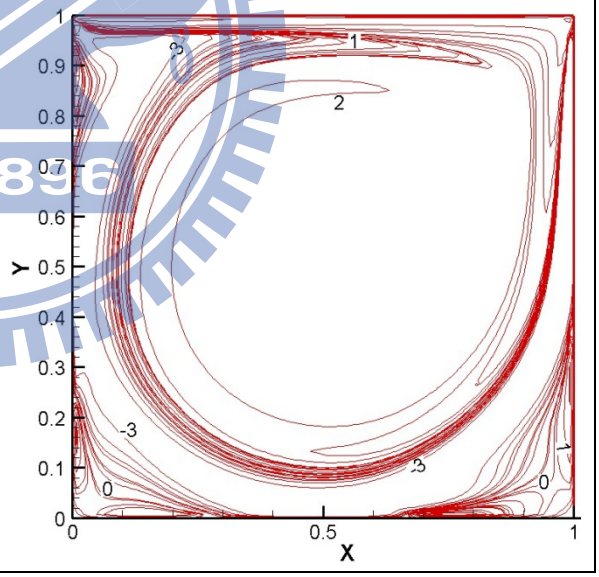
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 9 雷諾數 3200 之渦度比較圖

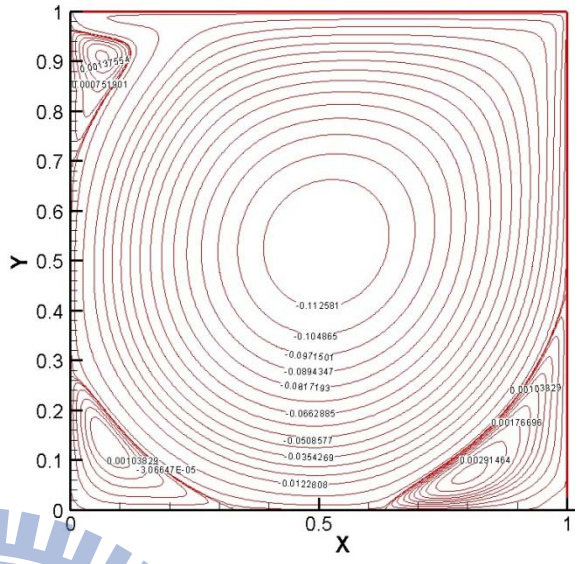
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 10 雷諾數 5000 之流線比較圖

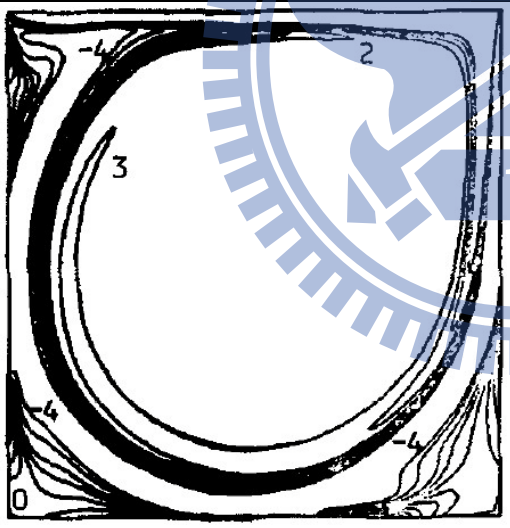
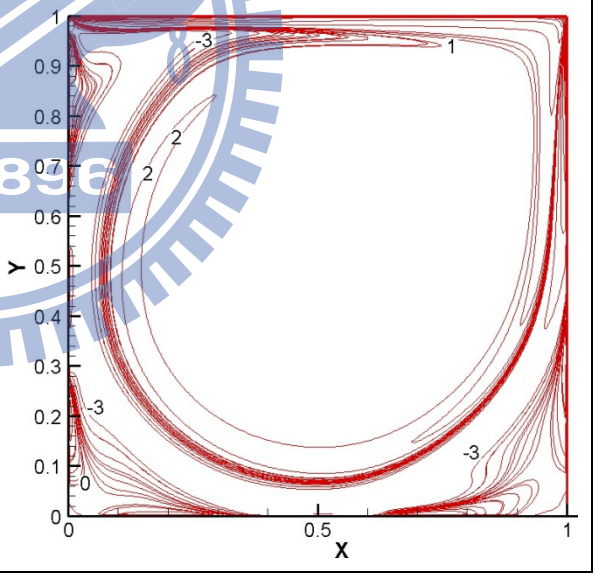
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 11 雷諾數 5000 之渦度比較圖

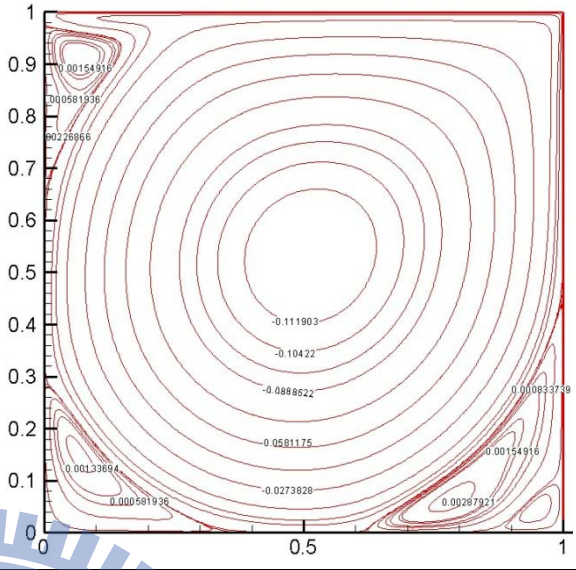
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 12 雷諾數 7500 之流線比較圖

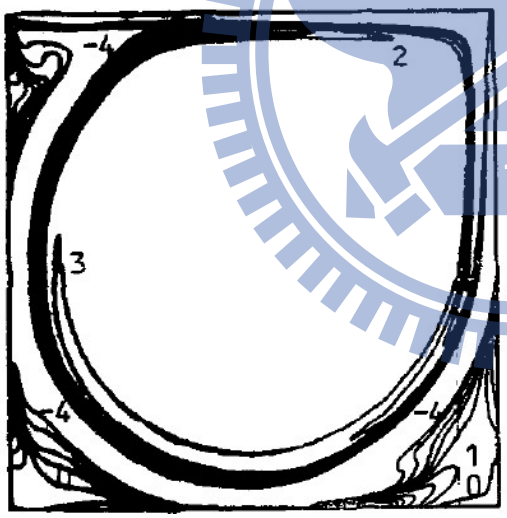
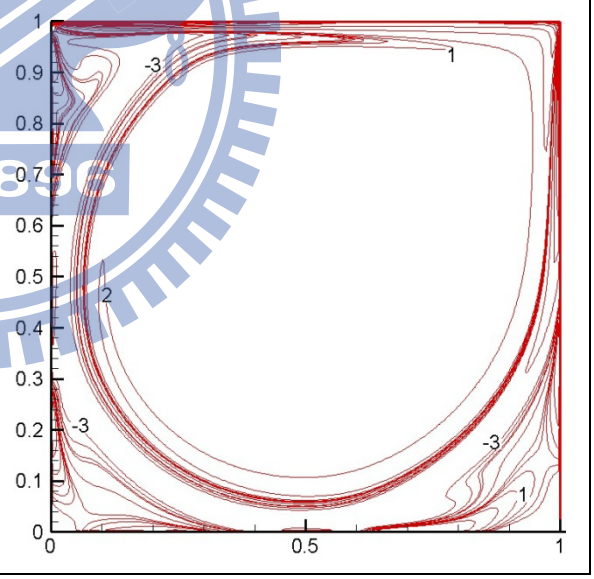
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 13 雷諾數 7500 之渦度比較圖

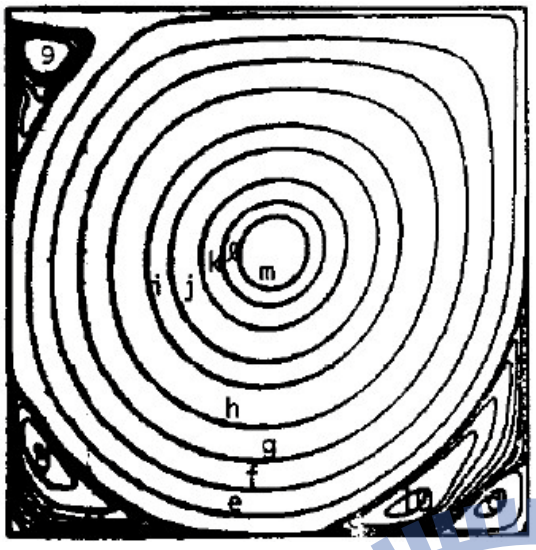
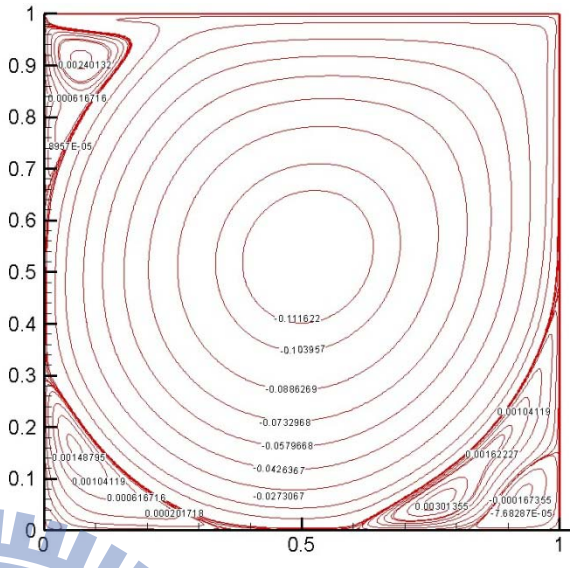
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 14 雷諾數 10000 之流線比較圖

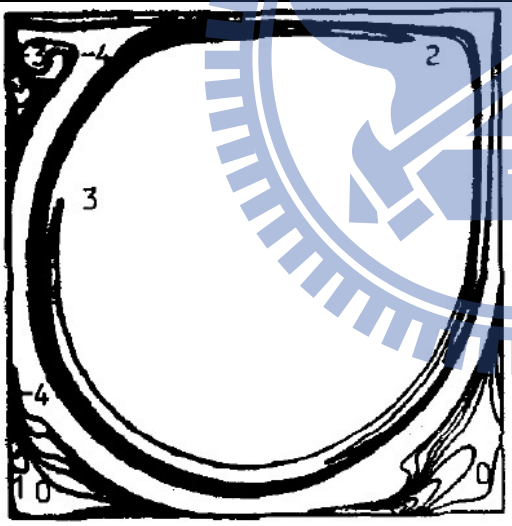
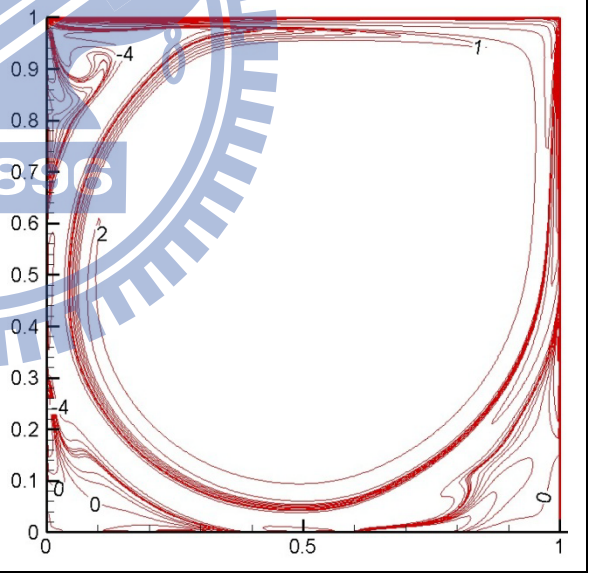
Ghia et al. (1982) 模擬結果	本研究模擬結果
	

圖 5- 15 雷諾數 10000 之渦度比較圖

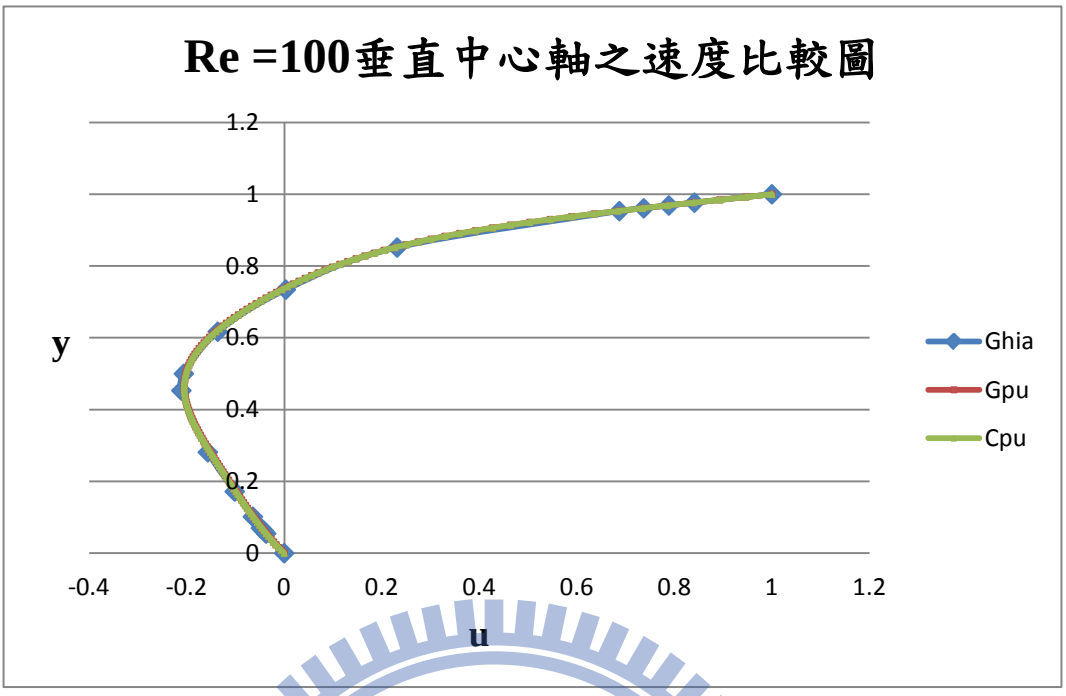


圖 5- 16 Re =100 垂直中心軸之速度比較圖

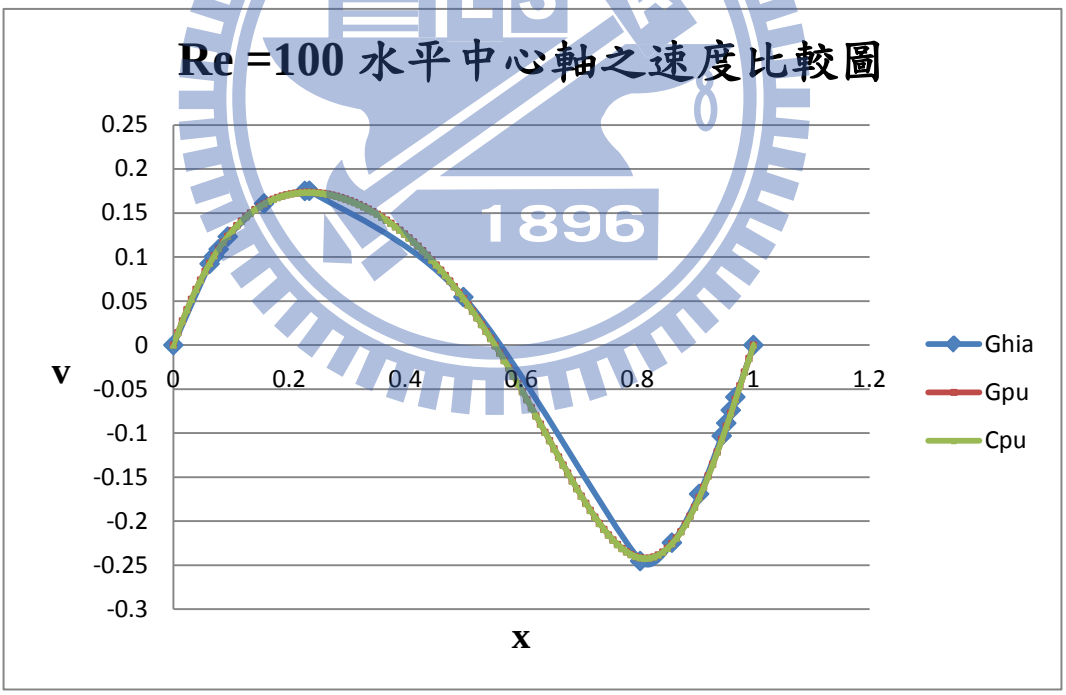


圖 5- 17 Re =100 水平中心軸之速度比較圖

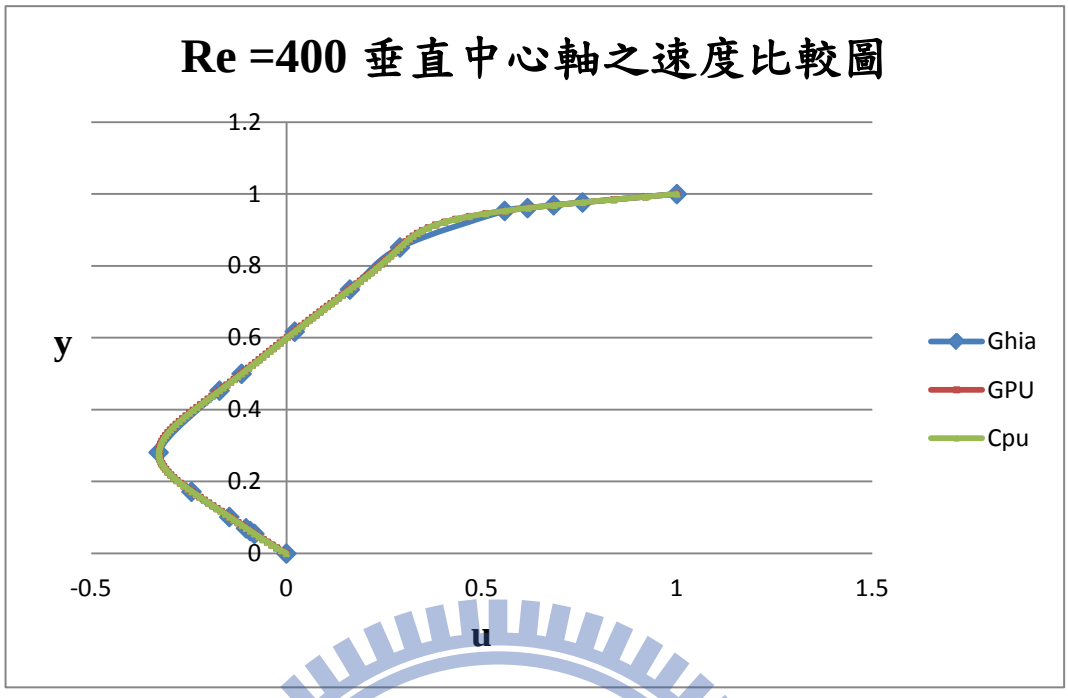


圖 5- 18 Re =400 垂直中心軸之速度比較圖

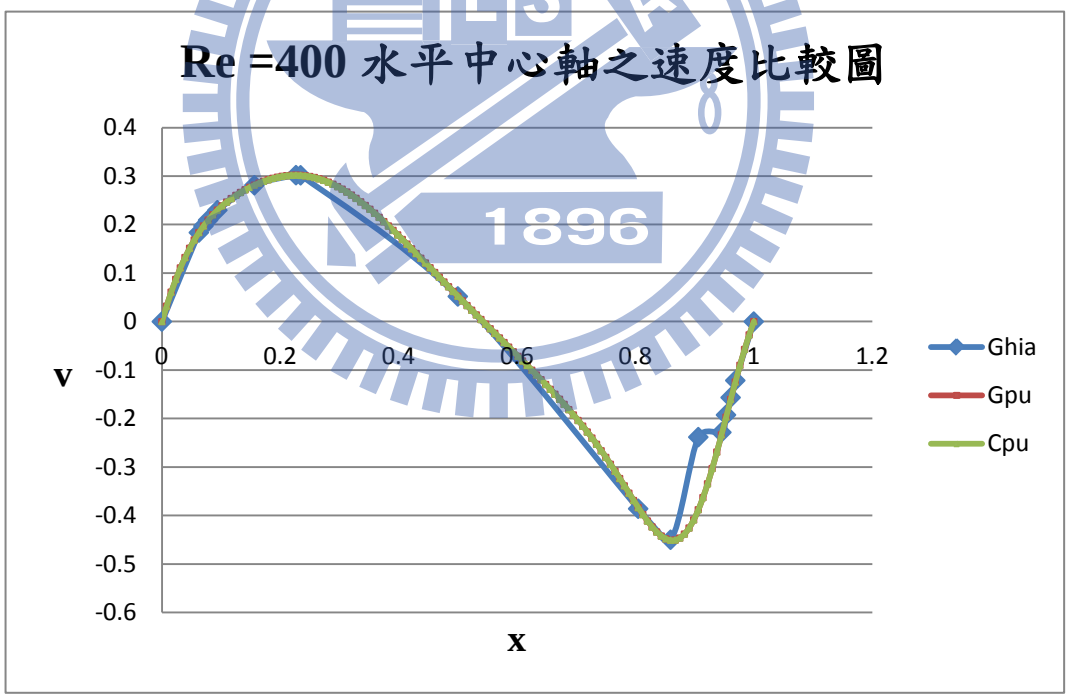


圖 5- 19 Re =400 水平中心軸之速度比較圖

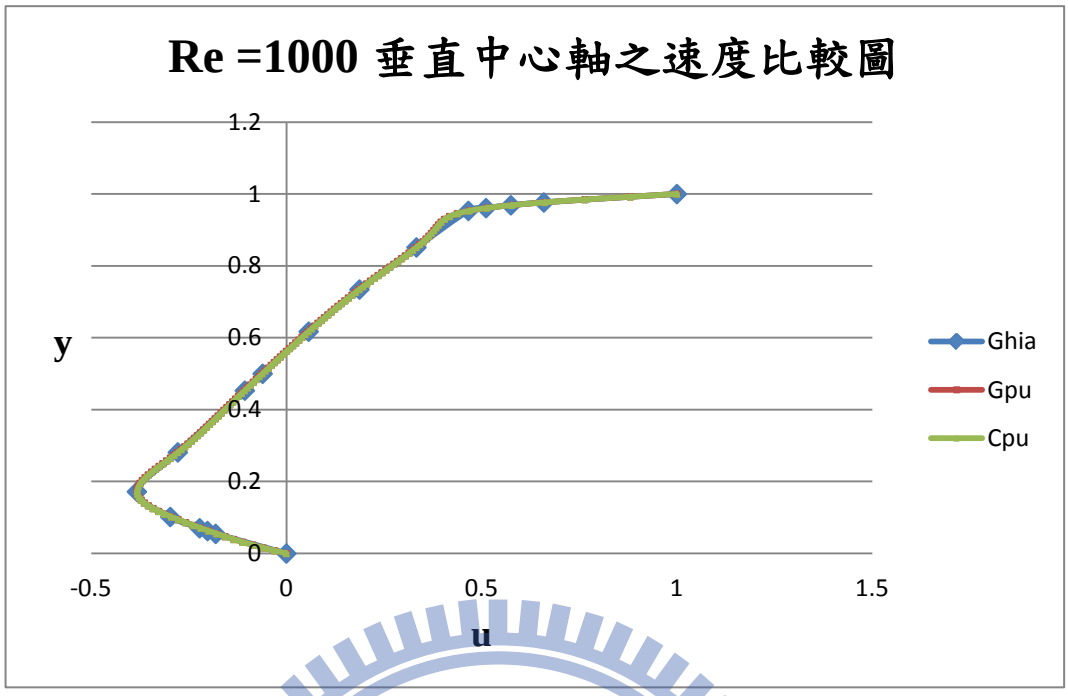


圖 5- 20 Re =1000 垂直中心軸之速度比較圖

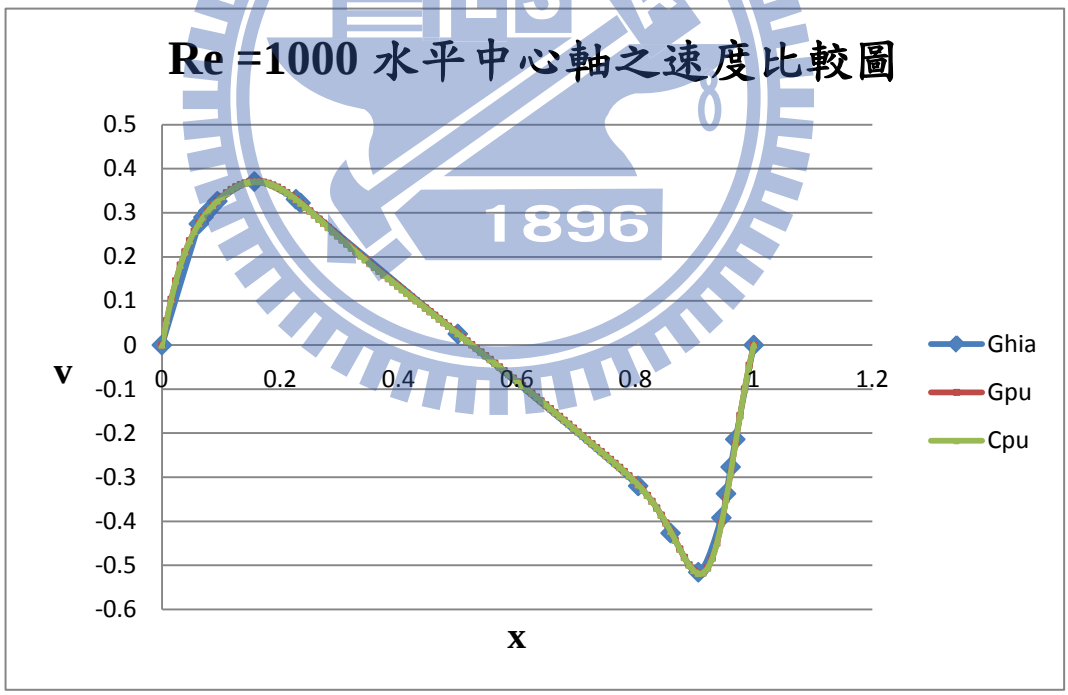


圖 5- 21 Re =1000 水平中心軸之速度比較圖

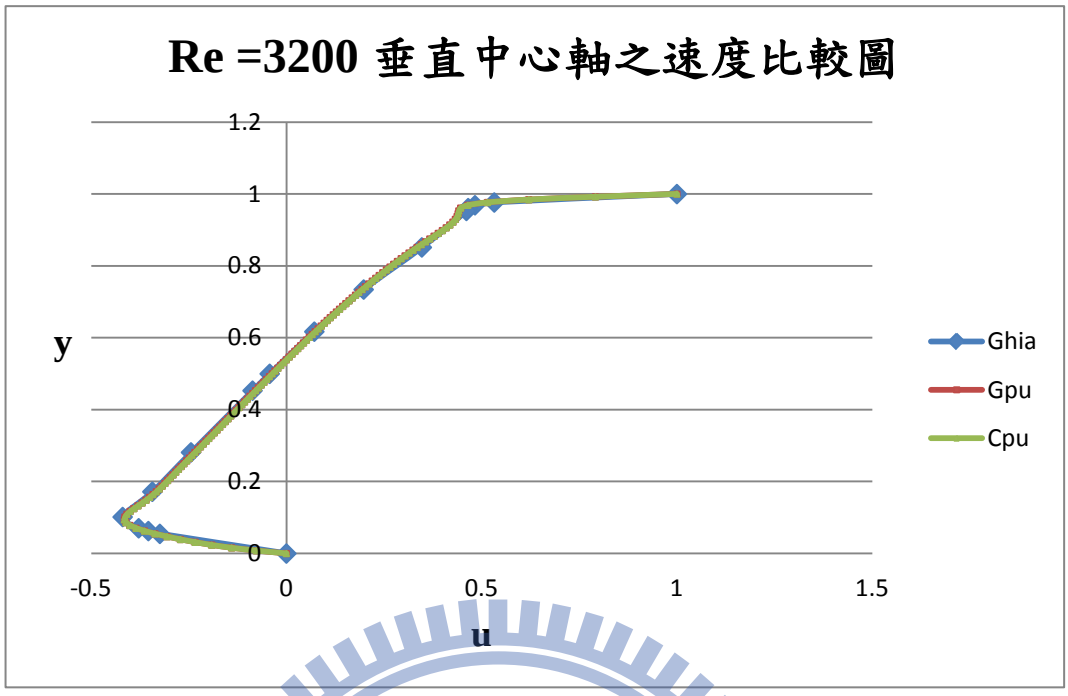


圖 5- 22 Re =3200 垂直中心軸之速度比較圖

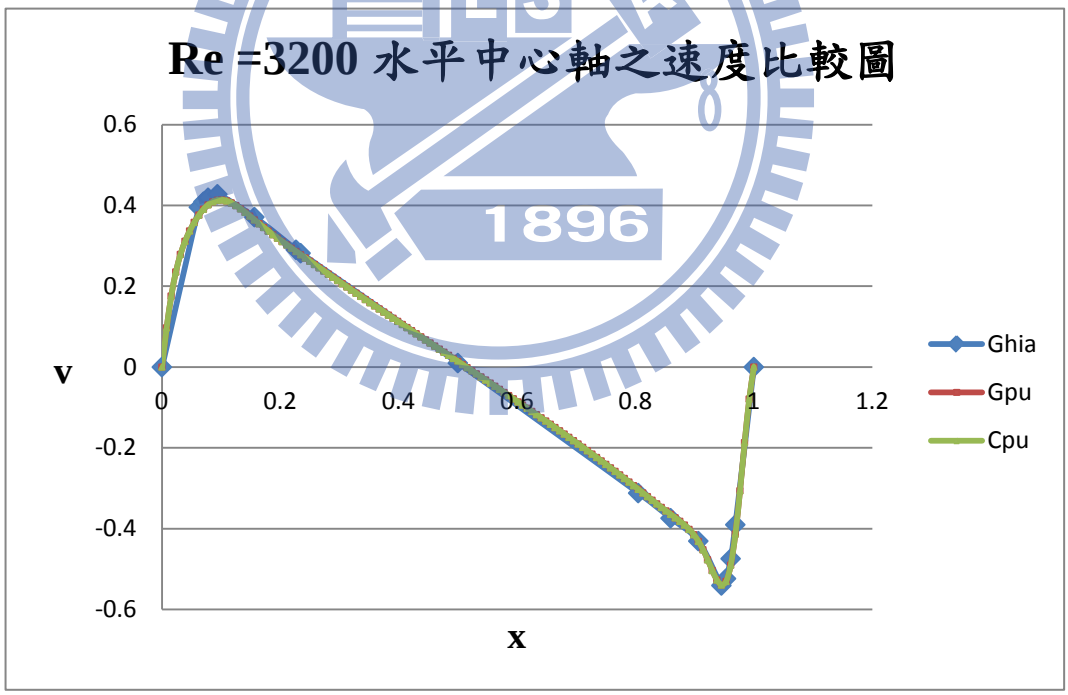


圖 5- 23 Re =3200 水平中心軸之速度比較圖

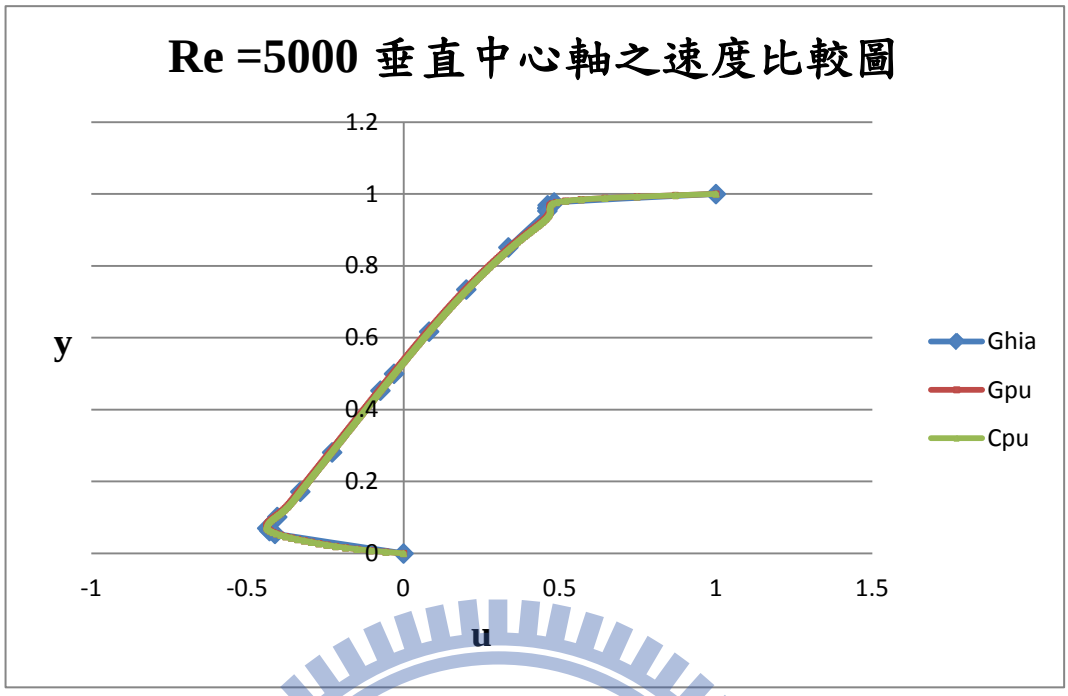


圖 5- 24 Re =5000 垂直中心軸之速度比較圖

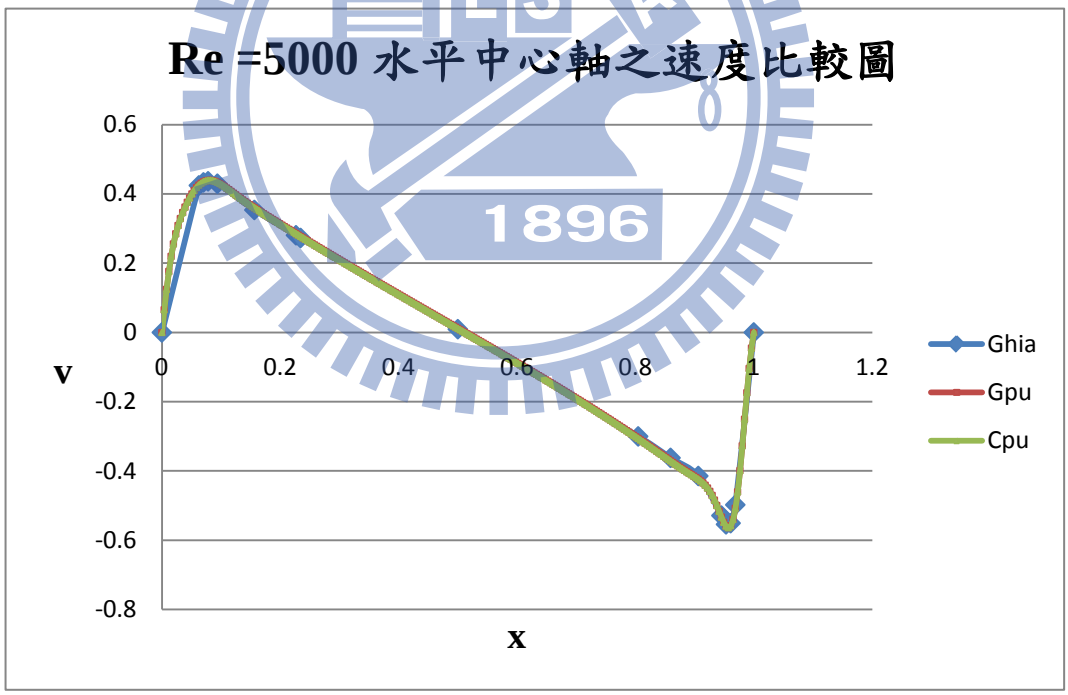


圖 5- 25 Re =5000 水平中心軸之速度比較圖

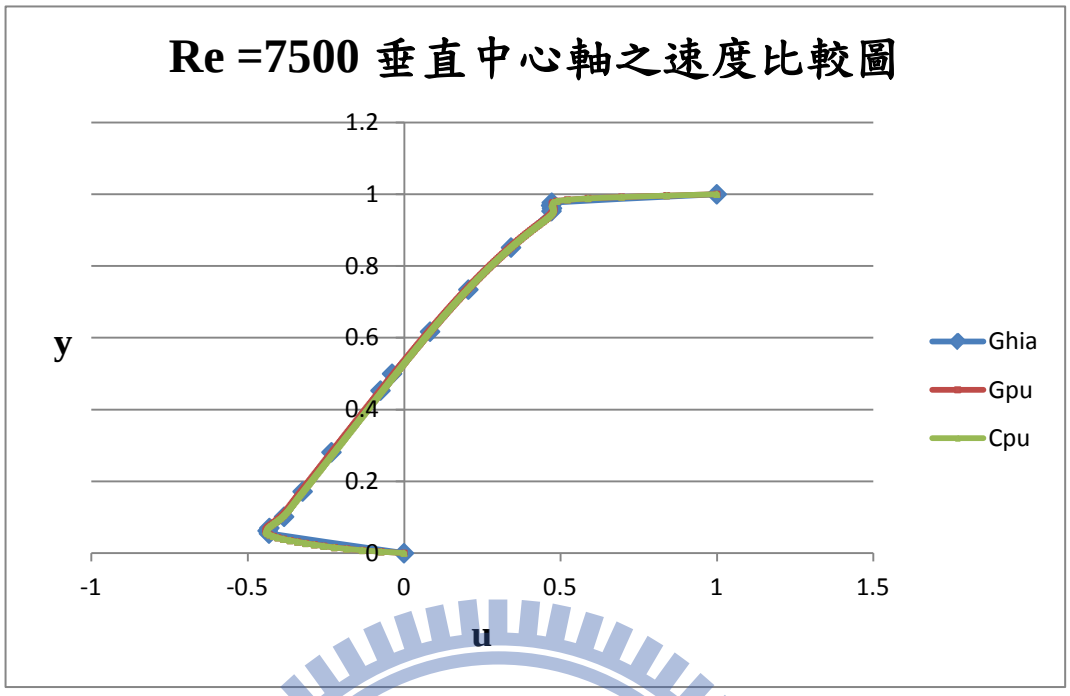


圖 5- 26 Re =7500 垂直中心軸之速度比較圖

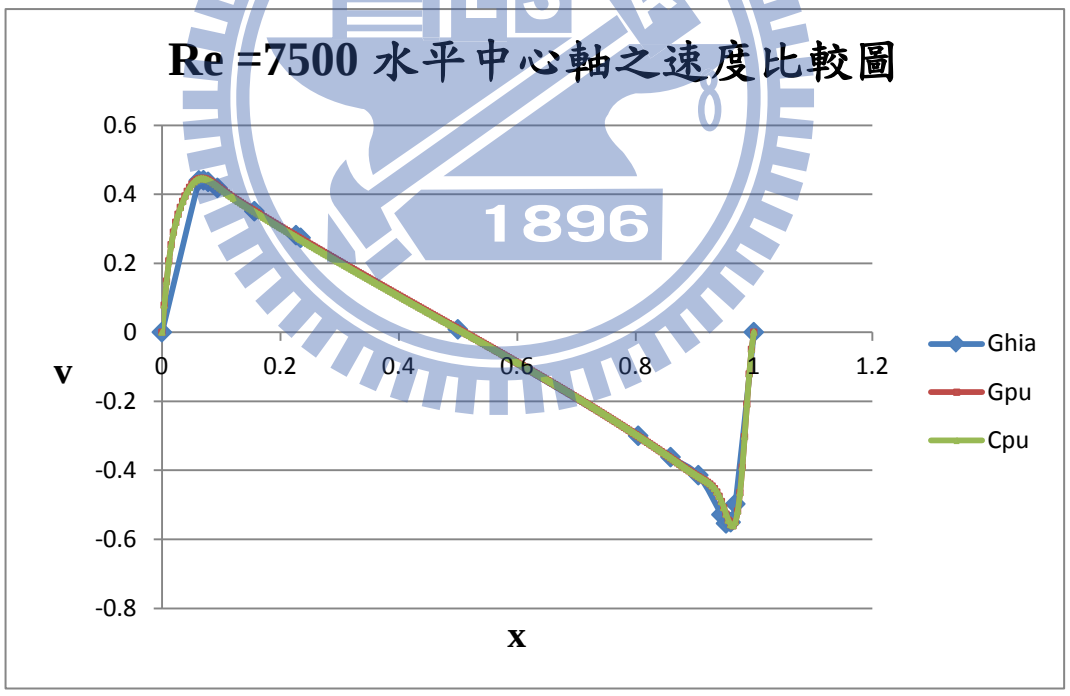


圖 5- 27 Re =7500 水平中心軸之速度比較圖

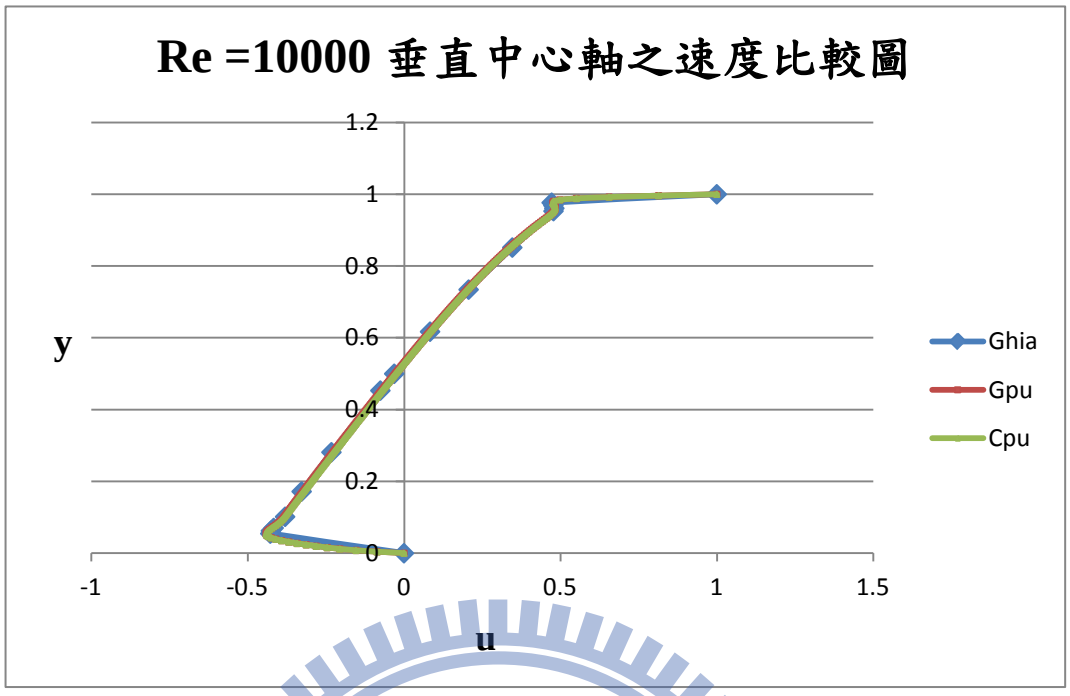


圖 5- 28 Re =10000 垂直中心軸之速度比較圖

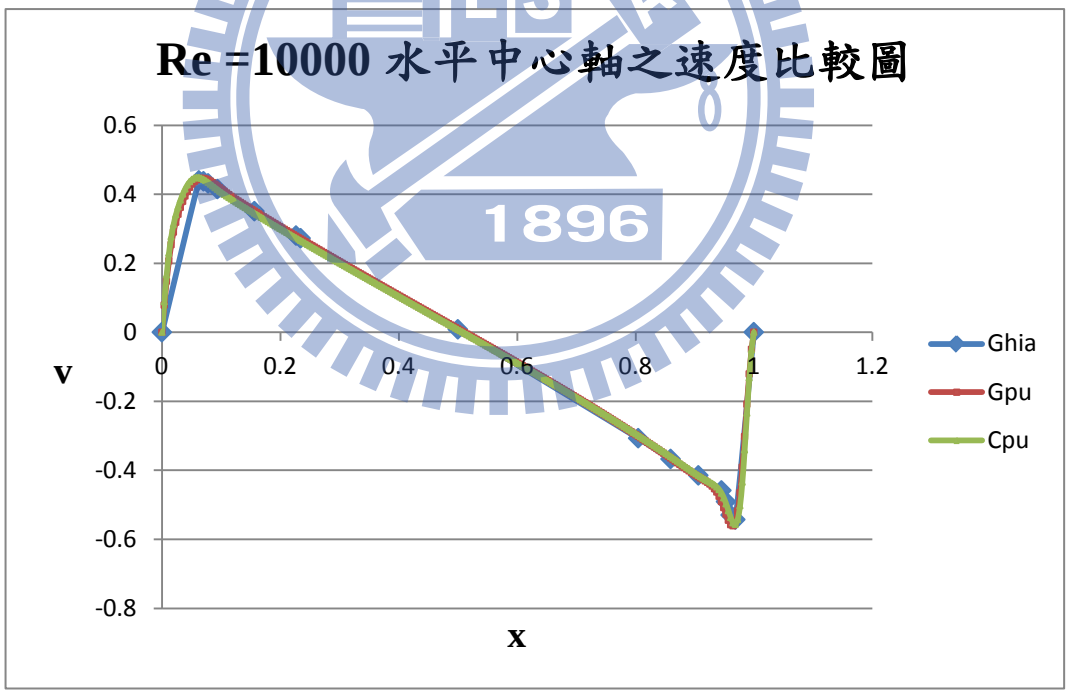


圖 5- 29 Re =10000 水平中心軸之速度比較圖

5.1.3 長型穴流之模擬

根據 Cheng (2006) 運用晶格波茲曼法（lattice Boltzmann method）在改變長寬比 D (D = H / W ，H 為穴流模擬範圍之深度、D 為穴流模擬範圍之寬度) 下模擬渦度及流線演變之情況，如圖 5-30 所示。

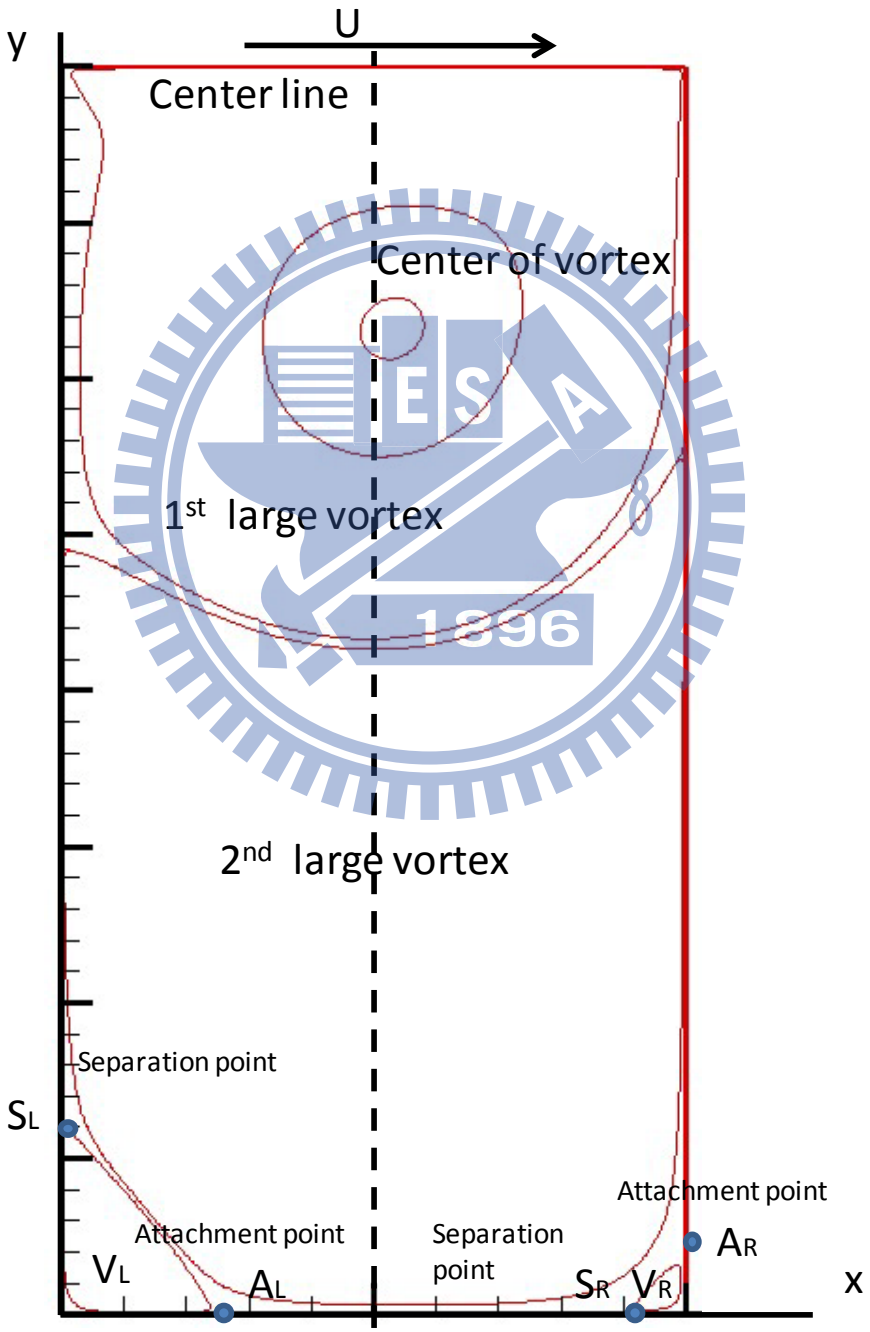


圖 5- 30 長型穴流示意圖

根據 Cheng (2006)模擬條件採用 201×201D (D 為穴流之長寬比)之非均勻網格，本研究則採用 201×(200×D+1)均勻網格下進行模擬。雷諾數固定為 1000 之下改變長寬比 D 分別為 1.1、1.15、1.2、1.25、2.2、2.3、2.4 及 3.2 進行模擬。在長寬比為 3.2 部分，採用 257×(256×D+1) 均勻網格下，因其受限於 GPU 平行運算之格網(grid)劃分所致，在模擬條件為 201×(200×D+1) 均勻網格下，運算將會發生錯誤，故而改用 257×(256×D+1)進行模擬（GPU 平行運算建議採用 2 的 N 次方做為格網單位，有利於其硬體架構演算。）模擬其流線、渦度與速度。模擬至其穩定狀態(steady state)，其流線模擬結果分別如圖 5-31 至圖 5-38 所示。

當 D = 1.1 時右下角的渦漩由於慣性的作用之下大於左下角之渦旋，如圖 5-31 所示。再將 D 提高至 1.15 左下角的附著點將與右下角的分離點共同組成一個停滯點，停滯點的位置並非恰好在其底面的中心位置如圖 5-32 所示。隨著 D 不斷的增加，停滯點的位置將會漸漸的脫離下壁面往上移，與下壁面左右兩側的渦旋合併如圖 5-33 所示，D 在由 1.2 增加至 1.25 時為其渦旋合併演變過程，在 D = 1.25 時第二個大渦旋已經合併生成如圖 5-34。隨著 D 不斷的增加，左下與右下又會產生一對新的渦旋，此時左下的渦旋會明顯的大於右下，在 D = 2.25 時第二個大渦旋會達到最大尺寸為 1.38，合併過程如圖 5-35~圖 5-36 所示，當 D = 2.4 時第三個大渦旋明顯產生如圖 5-37。當 D =3.2 時下壁面左右兩側又即將開始產生一對新的渦旋如圖 5-38

所示。由模擬結果可以發現流線的分布情況大致上相近，而在渦旋中心點位置大致上也符合 Cheng (2006) 模擬之結果。

隨後將長寬比 D 調整至 7 進行深長型的穴流模擬，調整雷諾數分別為 500、1000 及 5000 之下進行模擬，網格方面採用 257×1793 的均勻網格進行模擬。因此在模擬成果僅將流線函數結果取至10^{-7} 部分進行繪出，如圖 5-39 至圖 5-41 所示。Cheng (2006) 一文中提到深長型之穴流模擬極其耗時，並且在長寬比為 7.5 時流線函數結果已達10^{-15}，在穴流模擬範圍下壁面附近之網格必須切割得更為細緻才能提高其模擬可靠度，但是在加密其網格的過程中，運算量又更加龐大，模擬時間勢必增加許多。根據 IEEE-754 浮點數之定義，若採用雙倍精度儲存數字格式之下，有效數字可達 15 位，即精確度到小數點後第 14 位，因此在流線函數模擬結果亦有準確度問題存在，總體而言在進行極深之穴流數值模擬結果有準確性、模擬時間及可信度之問題。

在雷諾數為 500 之模擬，流線函數結果取至10^{-7}時僅能繪出三個渦旋，其渦旋中心點與 Cheng (2006)之研究成果相近，再提高雷諾數至 1000 及 5000 部分，便可繪製出四個渦旋的狀況，其渦旋中心點亦與 Cheng (2006) 之研究成果相近。

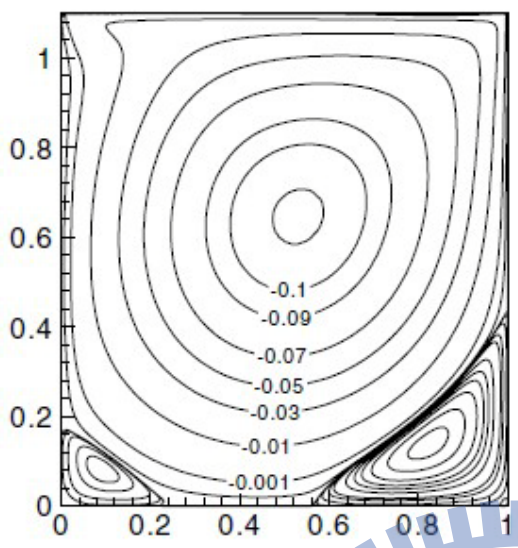
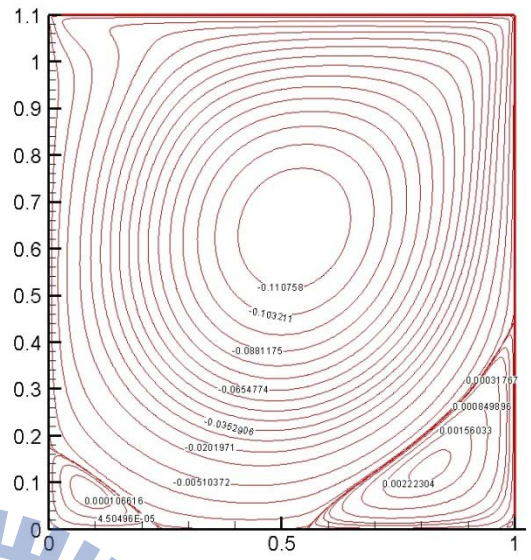
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 31 長寬比為 1 之流線比較圖

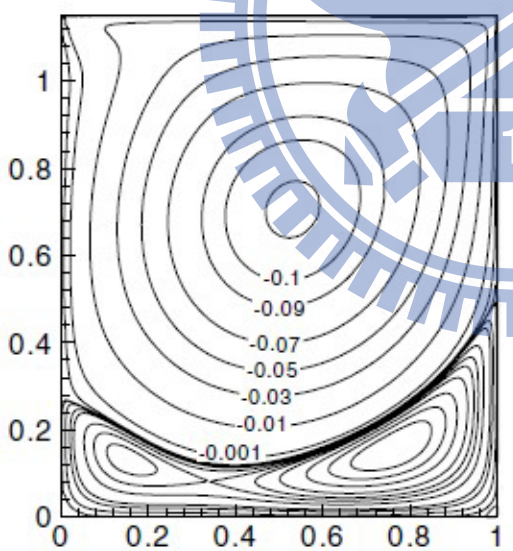
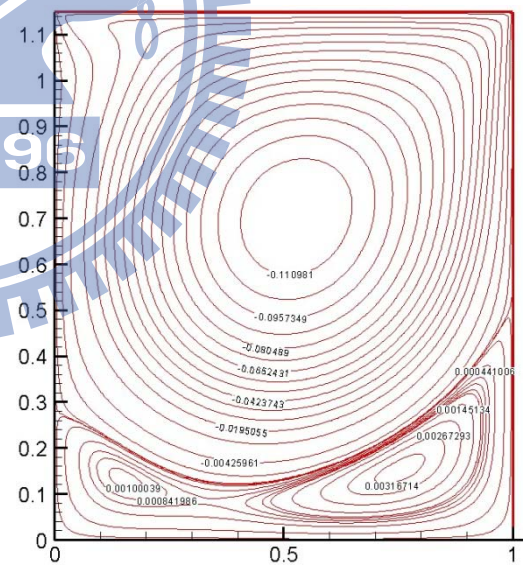
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 32 長寬比為 1.15 之流線比較圖

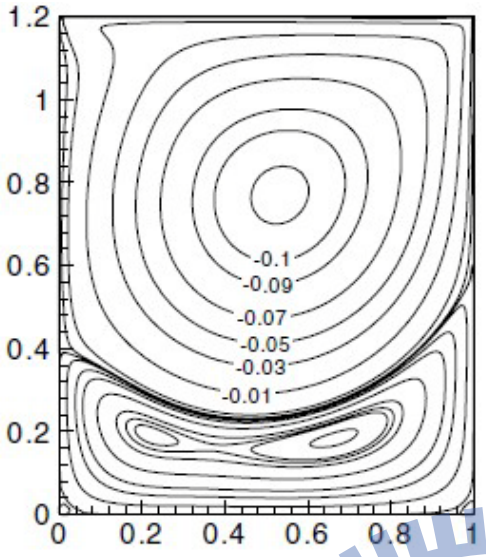
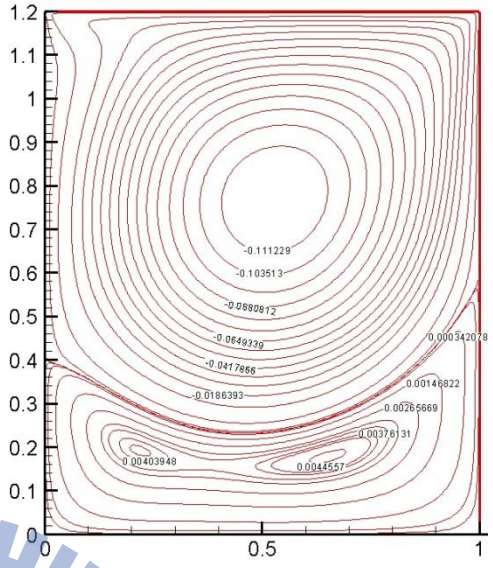
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 33 長寬比為 1.2 之流線比較圖

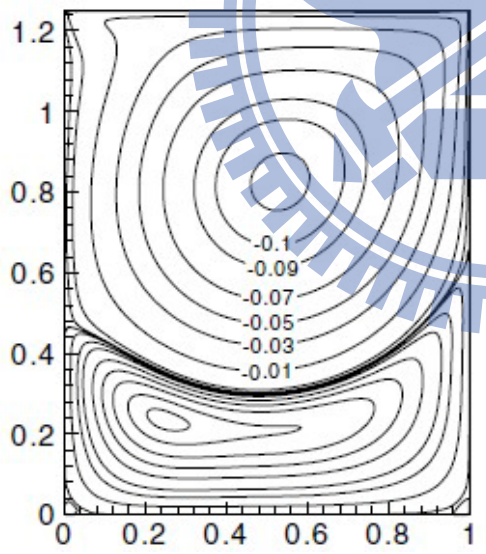
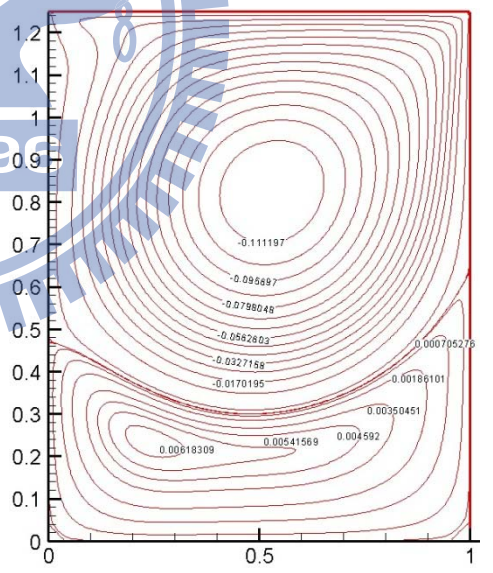
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 34 長寬比為 1.25 之流線比較圖

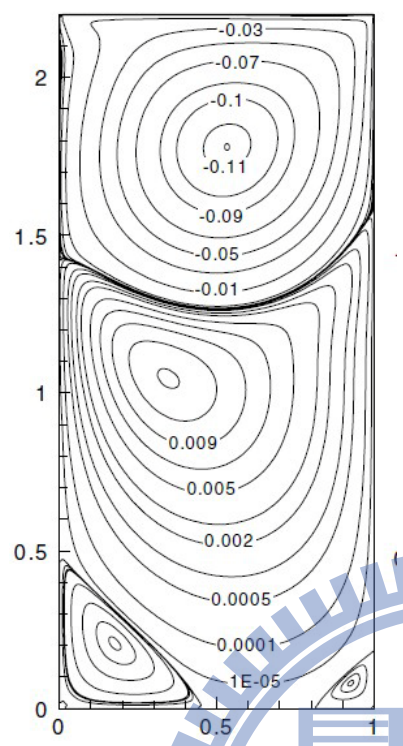
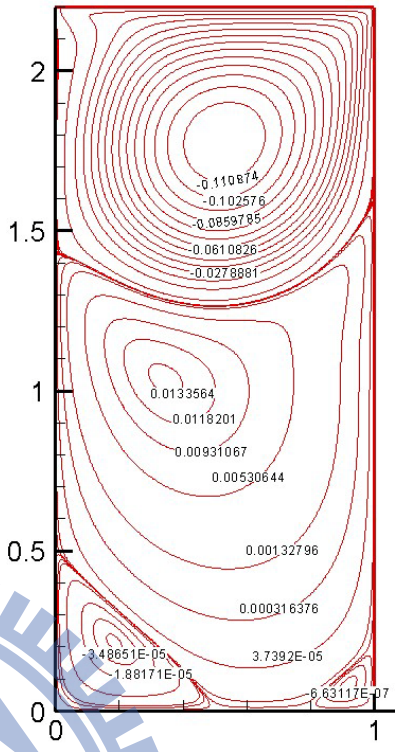
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 35 長寬比為 2.2 之流線比較圖

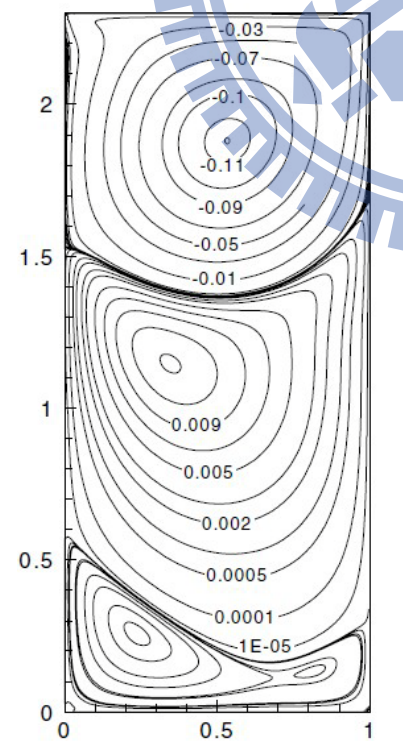
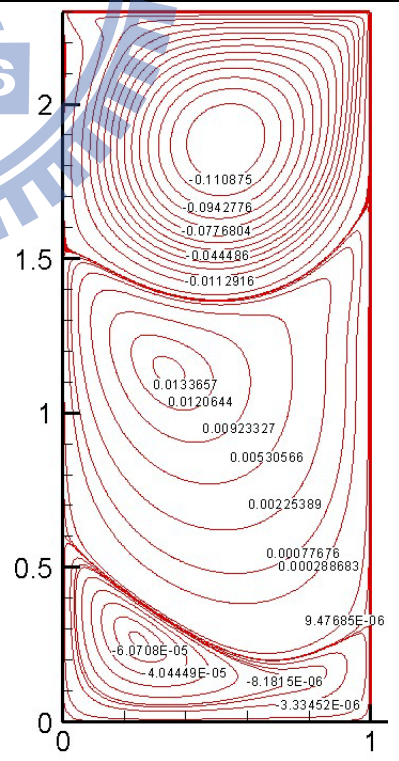
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 36 長寬比為 2.3 之流線比較圖

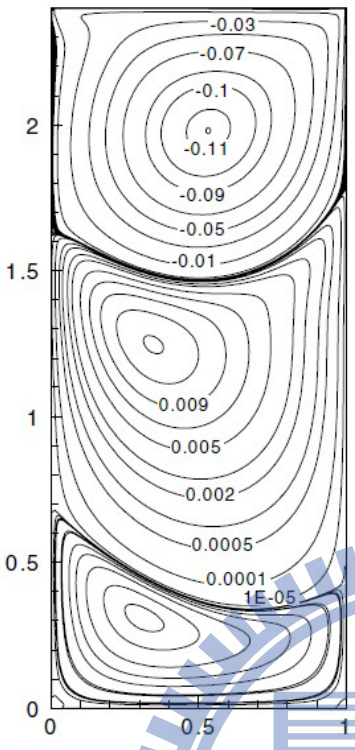
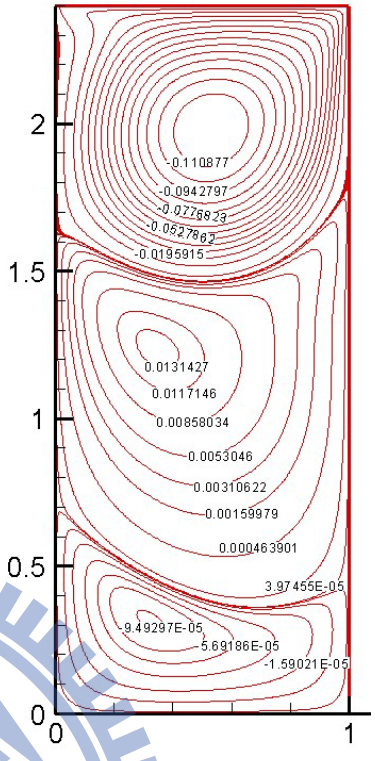
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 37 長寬比為 2.4 之流線比較圖

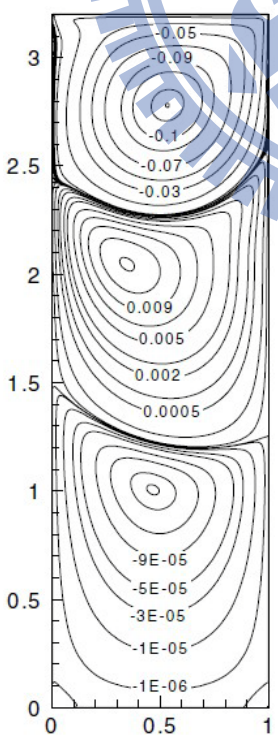
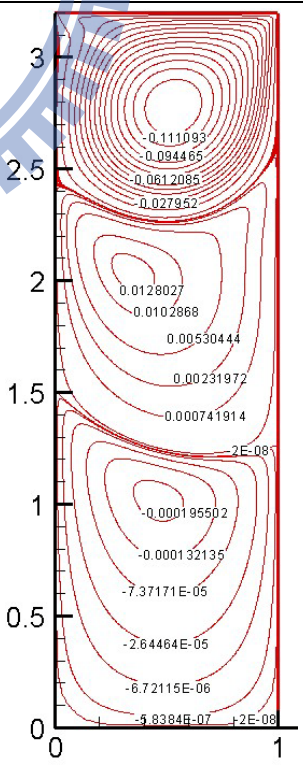
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 38 長寬比為 3.2 之流線比較圖

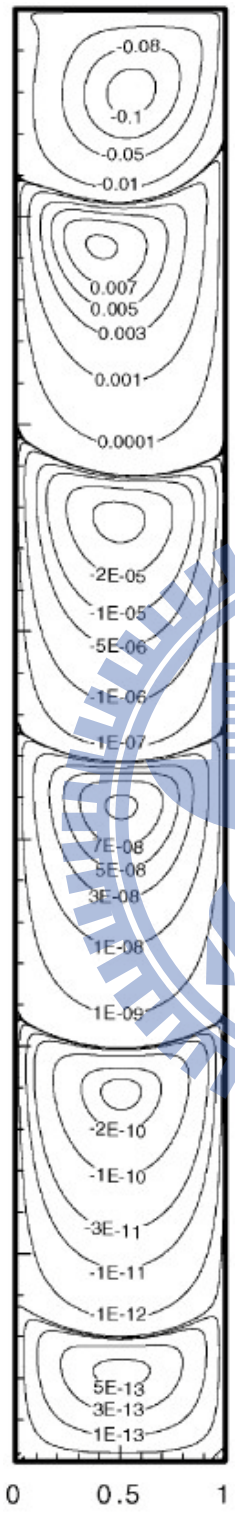
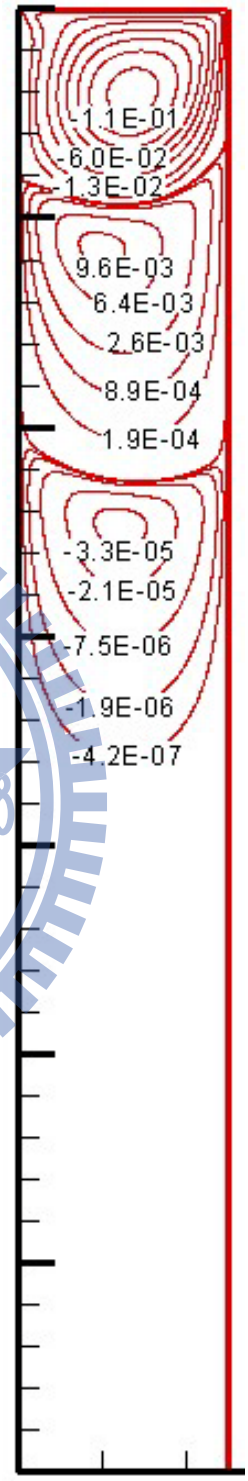
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 39 雷諾數 500 長寬比 7 之流線比較圖

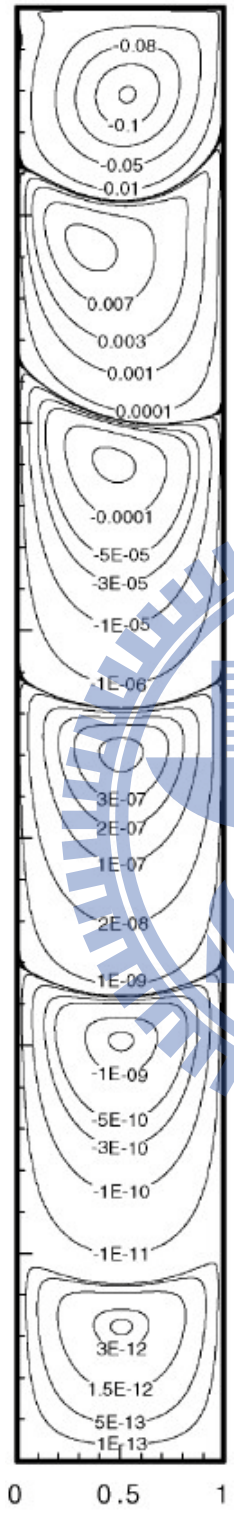
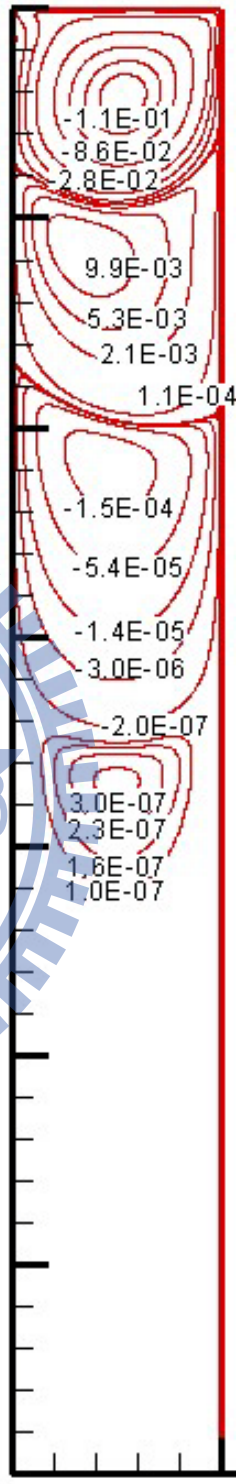
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 40 雷諾數 1000 長寬比 7 之流線比較圖

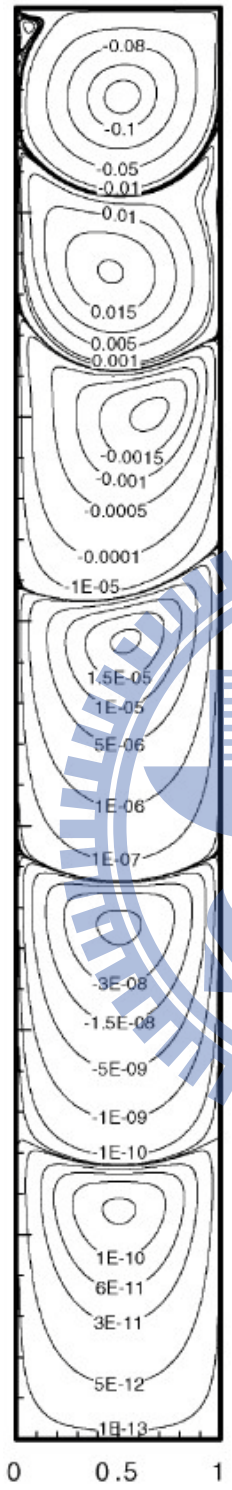
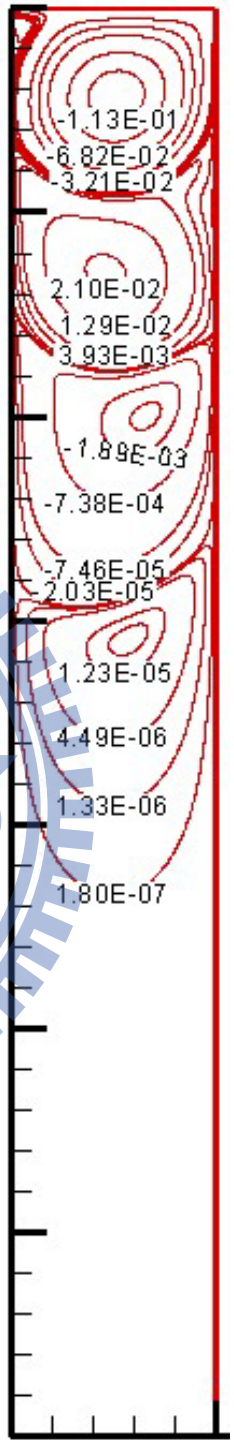
Cheng (2006) 模擬結果	本研究模擬結果
	

圖 5- 41 雷諾數 5000 長寬比 7 之流線比較圖

5.2 GPU 之加速成果與分析

針對穴流模擬條件在長寬比及雷諾數不同的狀況下，本研究在流線方程式部分採 Jacobi 迭代法進行運算，隨後在 CPU 運算程式碼加入了 PSOR (point successive over-relaxation)及 PGS (point gauss-seidel)兩種迭代收斂速度較快的演算法進行比較，對於程式模擬時間進行一系列加速評估。(本研究所採用時間為 CPU 運行時間，加速比定義：CPU 程式碼運行時間 / 改寫為 GPU 程式碼後運行時間)

在平行運算的架構下，所有運算單元必需處於同一時刻，因故 GPU 無法採用 PSOR 以及 PGS 此兩種演算法。

5.2.1 方穴流模擬評估

在改變雷諾數分別為 400、1000、3200 及 5000 下進行方穴流之模擬，模擬條件、完成時間與迭代次數如表 5-1　GPU–Jacobi 與 CPU–Jacobi 兩欄所示，在迭代次數方面採計渦度方程式迭代次數，透過圖 5-42 可以發現在雷諾數在 5000 並且網格數於 257×257 下，模擬所耗費時間有巨額的差異，透過圖 5-43 可知在網格大小為 129×129 之下總體加速度大約介於 3.5~4.5 倍之間，將網格大小擴大至 257×257 下總體模擬時間加速可以達到 9.4 倍左右，其收斂速度快慢會與模擬的起始條件有關，由此模擬可略窺到網格數量愈大所得到的平行計算的效益越高。

隨後改變 CPU 在流線方程式部分之演算法，採用收斂速度較快速的

PGS 及 PSOR 兩種迭代法進行模擬，在 PSOR 鬆弛系數方面選用 1.78，模擬條件、完成時間與迭代次數如表 5-1　CPU–PGS 與 CPU– PSOR 兩欄所示，透過表 5-1 可得知 PGS 與 PSOR 在求解流線方程式方面較良好的收斂速度，其中又以 PSOR 效率為最高，將總模擬時間縮短至 712 秒左右完成模擬，透過圖 5-44 可知，就算 CPU 方面採用收斂性較佳的數學運算式，而 GPU 仍然有顯著的加速效果，其加速效益在總模擬時間上仍可達到 2.7 倍左右。

由於 Jacobi、PGS 及 PSOR 在求解流線方程式時，收斂速度及所逼近之函數結果有所差異，導致渦度方程式總迭代次數有所差異，因此取迭代一次渦度方程式所花費之時間做為比較，加速比如圖 5-45 所示，可以發現到在網格大小相同時迭代所獲得之加速效益約略相等(8.2 倍~9.4 倍)，並且隨網格大小的擴大，加速效益也越來越顯著。

5.2.2 長型穴流模擬評估

在固定雷諾數為 1000 下，改變其長寬比 D 分別為 1.1、1.15、1.2、1.25、2.2、2.3、2.4 及 3.2 進行模擬進行，網格部分採用 201×(200×D+1)均勻網格下進行模擬，長寬比為 3.2 部分，則採用 257×(256×D+1) 均勻網格下，模擬條件、模擬完成時間與迭代次數如表 5-2 所示 GPU–Jacobi 與 CPU–Jacobi 兩欄所示，透過圖 5-46 可發現隨網格數量增加下，GPU 與 CPU 模擬運行時間也隨之增長，在長寬比達 3.2 之下 CPU 之運算時間比 GPU 多出 4000

秒左右，透過圖 5-47 可發現隨網格大小增加，加速成效越明顯，在長寬比 D 為 3.2 時網格數於 257×821 下加速成效可達 23 倍左右。

隨後改變 CPU 端於流線方程式部分之演算法，採用收斂速度較快速的 PGS 及 PSOR 兩種迭代法進行模擬，在 PSOR 鬆弛系數方面選用 1.78，模擬完成時間與迭代次數如表 5-2　CPU–PGS 與 CPU–PSOR 兩欄所示，透過表 5-2 可得知 PSOR 在求解流線方程式方面較良好的收斂效益，而 PGS 法在求解長型穴流之流線方程式時，收斂效益不如預期，其模擬時間比 Jacobi 來的緩慢，透過圖 5-48 可發現 CPU 方面採用收斂性較佳的數學運算式 GPU 仍然有顯著的加速效果，其加速效益在總模擬時間上仍可達到 21 倍左右。

由於 Jacobi、PGS 及 PSOR 在求解流線方程式時，在不同長寬比下收斂速度及所逼近之函數結果有所差異，導致渦度方程式總迭代次數有所差異，因此取迭代一次渦度方程式所花費之時間做為比較，加速比如圖 5-49 所示，隨著網格尺寸逐漸放大，更能看出平行演算之加速成效，在網格數為 257×821 下，CPU 與 GPU 迭帶一次所花費之時間可達 23 倍左右，與總模擬時間加速成效約略相等。

在長寬比固定為 7 設定其網格大小為 257×1793 下，改變其雷諾數分別為 500、1000 及 5000 下進行模擬，模擬完成時間與迭代次數如表 5-3 所示，在圖 5-50 可看出採用 PSOR 法進行計算時，時間相較於 Jacobi 法與 PGS 法而言要短的許多，在雷諾數為 5000 時模擬時間約略短少了 30000 秒左右，

但是透過 GPU 平行化演算進一步加速能再將時間縮短 30000 秒，由圖 5-51 可看出雷諾數為 5000 下，GPU 端流線運算式與 CPU 端皆使用 Jacobi 法時，以總模擬時間做為比較時有 22 倍左右加速效果，在雷諾數為 1000 下則可高達 33 倍加速效果；若 GPU 端採用 Jacobi 法時相對於 CPU 端使用收斂性較佳之 PSOR 法而言以總模擬時間做為比較，則有 12.7 倍的加速效果，在雷諾數為 1000 下則有 27 倍左右的加速成效。

由於 Jacobi、PGS 及 PSOR 在求解流線方程式時，在不同雷諾數下收斂速度及所逼近之函數結果有所差異，導致渦度方程式總迭代次數有所差異，因此取迭代一次渦度方程式所花費之時間做為比較，加速比如圖 5-52 所示，在雷諾數越高的情況下欲使流線方程式達到收斂的時間會越長，導致在雷諾數為 5000 時加速效果較雷諾數為 1000 下低落，就其在雷諾數為 1000 下相較於 Jacobi 算法有 33 倍左右加速效果，而 PSOR 亦達到了 27 倍左右，與總時間模擬加速成效約略相同。

5.2.3 雷諾數固定下改變網格大小

透過固定雷諾數為 1000 改變網格大小情況下，探討 GPU 加速之效益，在 CPU 端分別採用了 Intel® Core™2 Duo Processor E7400 (3M Cache, 2.80 GHz, 1066 MHz FSB) 與 AMD Athlon II X4 635 (L1 512KB, L2 1MB, 2.9GHz)進行比較，流線函數部分演算法在 GPU 與 CPU 上均採用 Jacobi 法，模擬條件及成果如表 5-4 所示，透過圖 5-53 可知在網格數量小

時，AMD 與 Intel 之運算時間差異較小，而網格一經擴大兩者在運算時間上便開始有顯著差異，一般而言 Intel 公司所開發之中央處理器對於編譯器有最佳化的效果，並且其 CPU 內之指令集架構較有利於處理數值運算部分，在網格數達 257×1793 下，模擬總時間大約差了 3600 秒，根據圖 5-54 可發現，在網格數達 257×1793 下，採用 AMD 設備模擬時間與 GPU 之平行運算時間相差了大約 44 倍，採用 Intel 設備則相差了 34 倍左右。

隨後採取迭代一次渦度方程式所花費之時間做為比較，如圖 5-55 所示，網格數量越大下，加速成效越顯著，網格數達 257×1793，以 AMD 之設備做為基準有 44 倍左右加速效果，Intel 則有 34 倍左右加速效果，與總時間加速成效約略相同。

在本次模擬條件雷諾數為 1000 下分別選定網格數大小為 129×129、257×257 及 513×513 進行方穴流模擬，其流線分布情形如圖 5-56 至圖 5-58 所示，放大左下與右下之渦旋進行觀察，可發現網格大小於 129×129 時，左側渦旋的邊緣銳化的相當嚴重，右側渦旋在尖峰處銳化情形較為明顯；隨後將網格大小放大至 257×257，左側與右側渦旋有顯著改善，但是在尖峰處仍有細微的銳化現象；將網格大小取至 513×513 時，左側與右側之渦旋之流線分布情況顯得平滑許多，僅左下與右下之極小渦旋仍然有銳化現象，推測是網格大小仍不夠細緻所導致。根據 Ghia et al. (1982)之水平與垂直中心線上速度計算結果，做為比較之基準如圖 5-59 至圖 5-60 所示，其 U、V

之速度在網格情況數不同下所求得之數值約略相同。

在選定網格大小為 129×129、257×257 及 513×513 進行模擬時，隨著網格點數增加，模擬時間也會隨之拉長，根據圖 5-61 可發現在採用網格大小 257×257 GPU 平行運算模擬之時間仍短於 CPU 在網格大小 129×129 下之模擬時間；網格大小為 513×513 GPU 平行運算模擬之時間仍短於 CPU 在網格大小 257×257 下之模擬時間，這意味著在耗費相同的模擬時間下，GPU 可進行較為細緻之模擬，網格點數較多下所得之流線相對平滑，以利探討複雜流場之細部情況。

表 5-1 方穴流模擬成果表

雷諾數	400	1000	3200	5000
網格大小	129×129	129×129	129×129	257×257
GPU - Jacobi				
迭代次數	91223	102223	148463	894307
模擬時間(秒)	11.969	13.5	19.531	268.594
迭代一次時間(秒)	0.000131	0.000132	0.000132	0.0003
GPU - Jacobi				
迭代次數	91681	100190	143341	891233
模擬時間(秒)	47.047	59.766	65.375	2516.578
總模擬時間加速比	3.93	4.43	3.35	9.37
迭代一次時間(秒)	0.000513	0.000597	0.000456	0.002824
迭代一次加速比	3.91	4.52	3.47	9.40
CPU - PGS				
迭代次數	91681	100190	143341	891233
模擬時間(秒)	42.422	46.016	65.25	2195.203
總模擬時間加速比	3.54	3.41	3.34	8.17
迭代一次時間(秒)	0.000463	0.000459	0.000455	0.002463
迭代一次加速比	3.53	3.48	3.46	8.20
CPU - PSOR				
迭代次數	89018	53321	72022	268665
模擬時間(秒)	42.609	25.218	34.078	712.36
總模擬時間加速比	3.56	1.87	1.74	2.65
迭代一次時間(秒)	0.000479	0.000473	0.000473	0.002651
迭代一次加速比	3.65	3.58	3.60	8.83

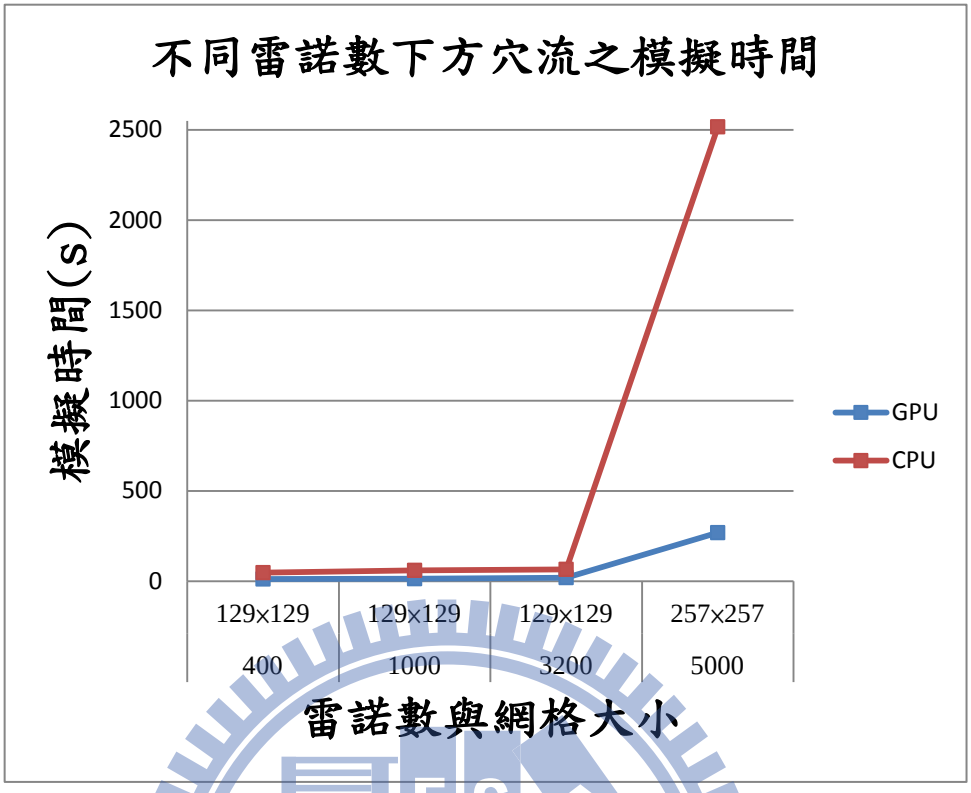


圖 5- 42 不同雷諾數下方穴流之總模擬時間

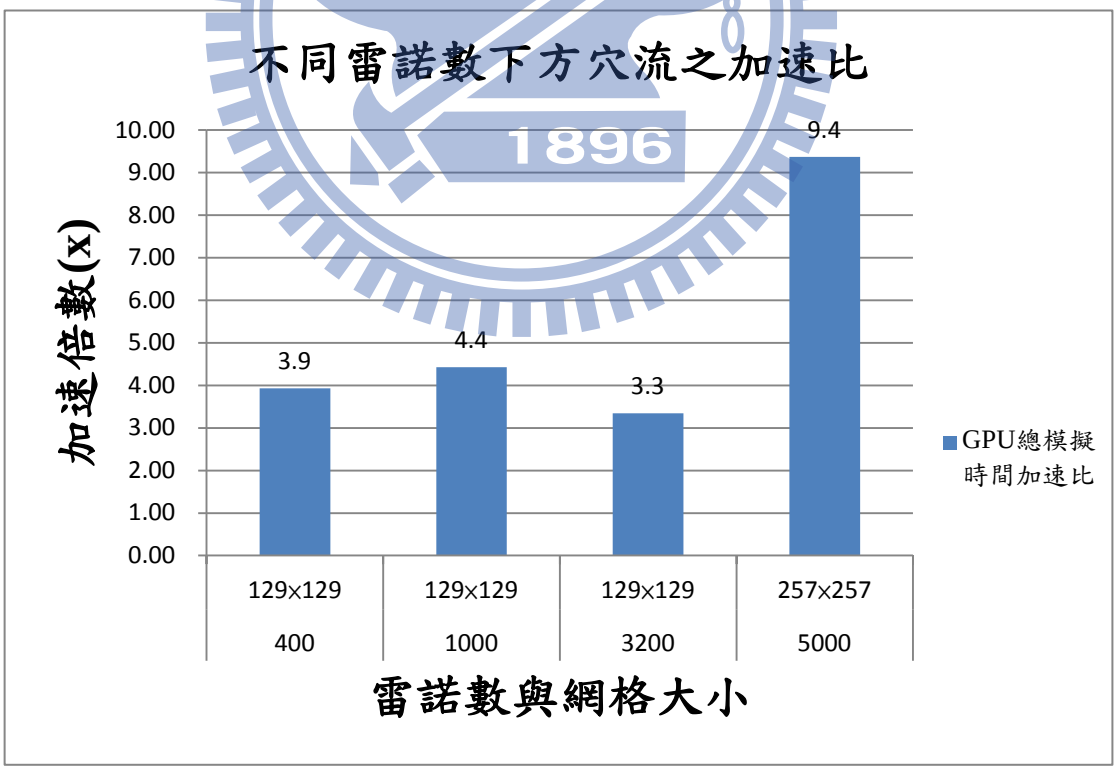


圖 5- 43 不同雷諾數下方穴流之加速比 1

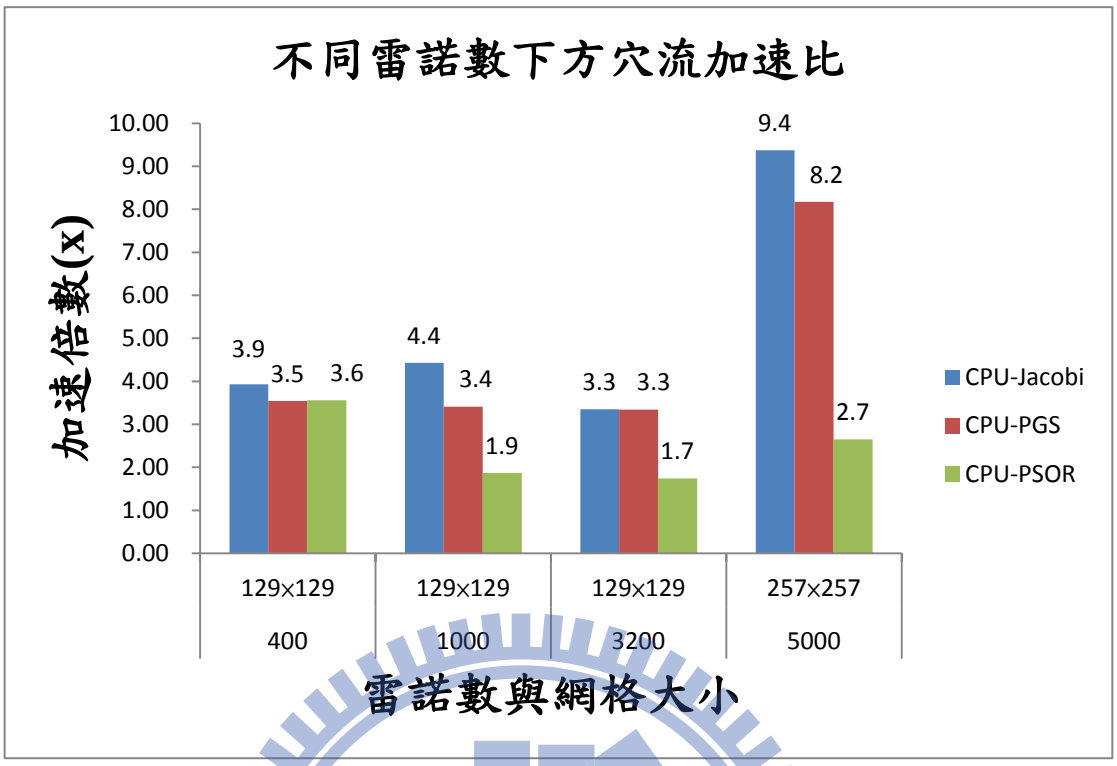


圖 5- 44 不同雷諾數下方穴流之加速比 2

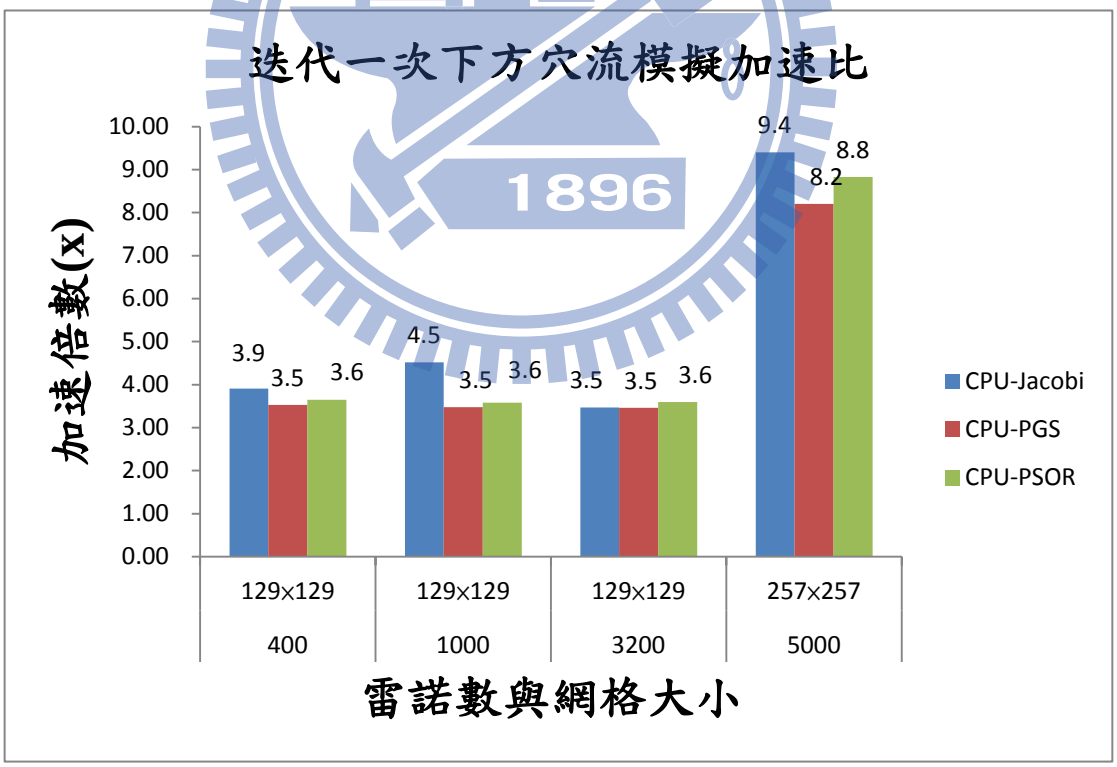


圖 5- 45 迭代一次下方穴流模擬加速比

表 5-2 改變長寬比穴流模擬成果表

長寬比 D = H/W	1.1	1.15	1.2	1.25	2.2	2.3	2.4	3.2
網格大小	201×221	201×231	201×241	201×251	201×441	201×461	201×481	257×821
GPU - Jacobi								
迭代次數	187749	254839	185918	167387	187331	186115	183889	230874
模擬時間(秒)	41.844	60.218	44.469	41.859	72	75.078	74.75	186.406
迭代一次時間(秒)	0.000223	0.000236	0.000239	0.00025	0.000384	0.000403	0.000406	0.000807
CPU - Jacobi								
迭代次數	189271	260290	194435	175854	184410	183862	183482	225736
模擬時間(秒)	415.015	590.578	512.266	443.875	848.11	948.328	1002.938	4223.843
總模擬時間加速比	9.92	9.81	11.52	10.60	11.78	12.63	13.42	22.66
迭代一次時間(秒)	0.002193	0.002269	0.002635	0.002524	0.004599	0.005158	0.005466	0.018711
迭代一次加速比	9.84	9.60	11.02	10.09	11.97	12.79	13.45	23.18
CPU - PGS								
迭代次數	189271	260290	194435	175854	184410	183862	183482	225736
模擬時間(秒)	376.656	581.14	532.109	452.75	872.344	988.797	1005.297	4558.641
總模擬時間加速比	9.00	9.65	11.97	10.82	12.12	13.17	13.45	24.46
迭代一次時間(秒)	0.00199	0.002233	0.002737	0.002575	0.00473	0.005378	0.005479	0.020195
迭代一次加速比	8.93	9.45	11.44	10.30	12.31	13.33	13.48	25.01
CPU - PSOR								
迭代次數	139398	159821	103132	60314	146135	145363	145838	199651
模擬時間(S)	300.5	352.782	269.328	149.328	715.25	744.625	712.859	3851.406
總模擬時間加速比	7.18	5.86	6.06	3.57	9.93	9.92	9.54	20.66
迭代一次時間(秒)	0.002156	0.002207	0.002611	0.002476	0.004894	0.005123	0.004888	0.019291
迭代一次加速比	9.67	9.34	10.92	9.90	12.73	12.70	12.02	23.89

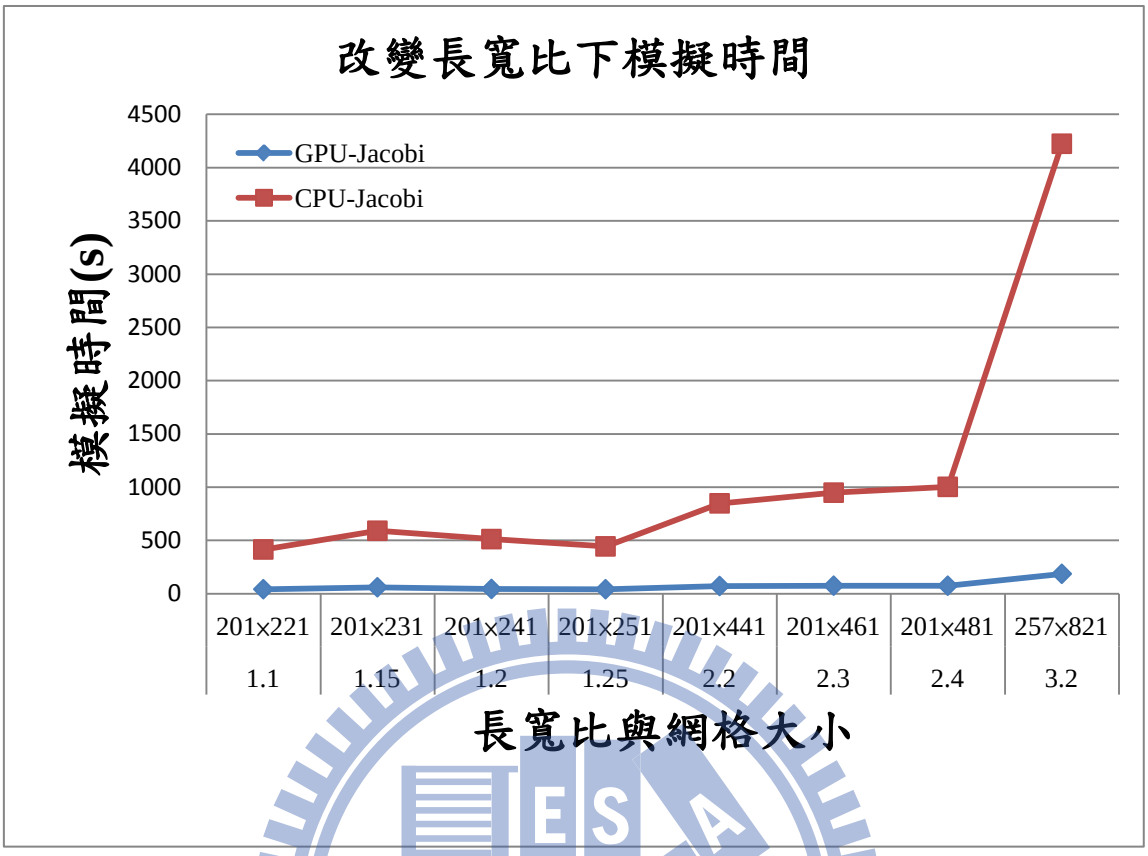


圖 5- 46 不同長寬比下穴流之總模擬時間

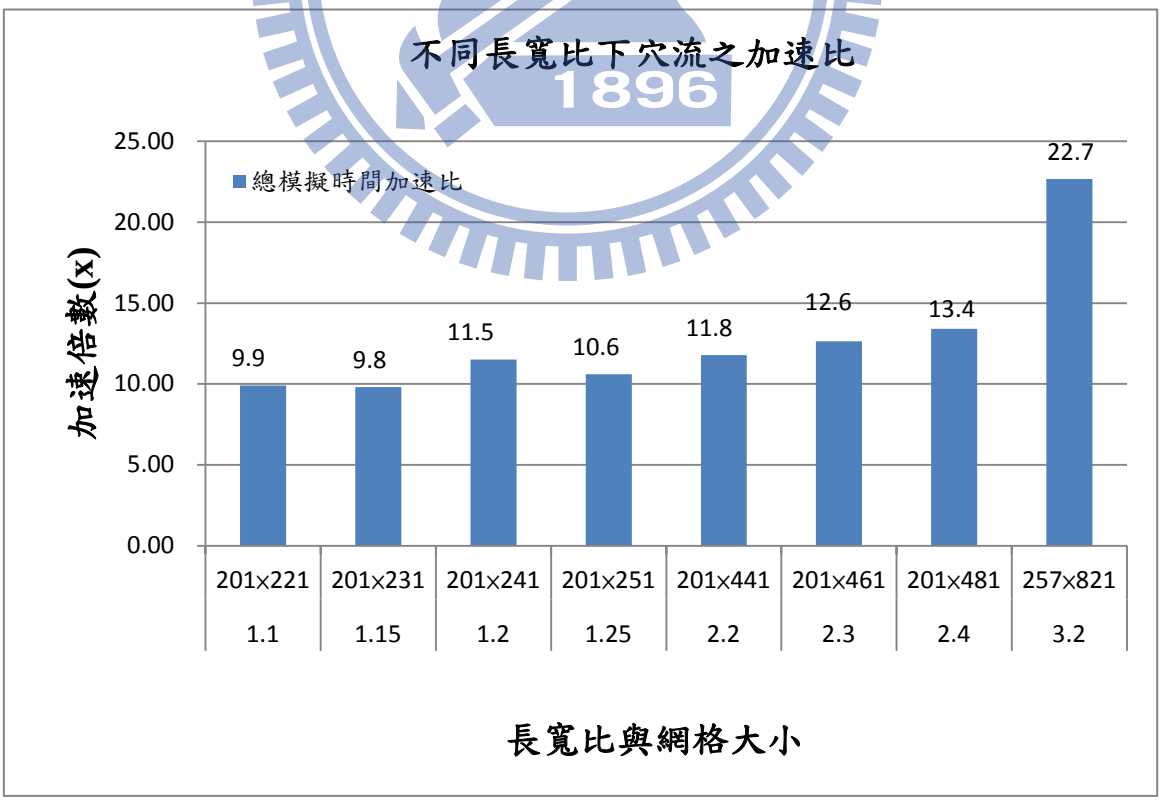


圖 5- 47 不同長寬比下穴流之加速比 1

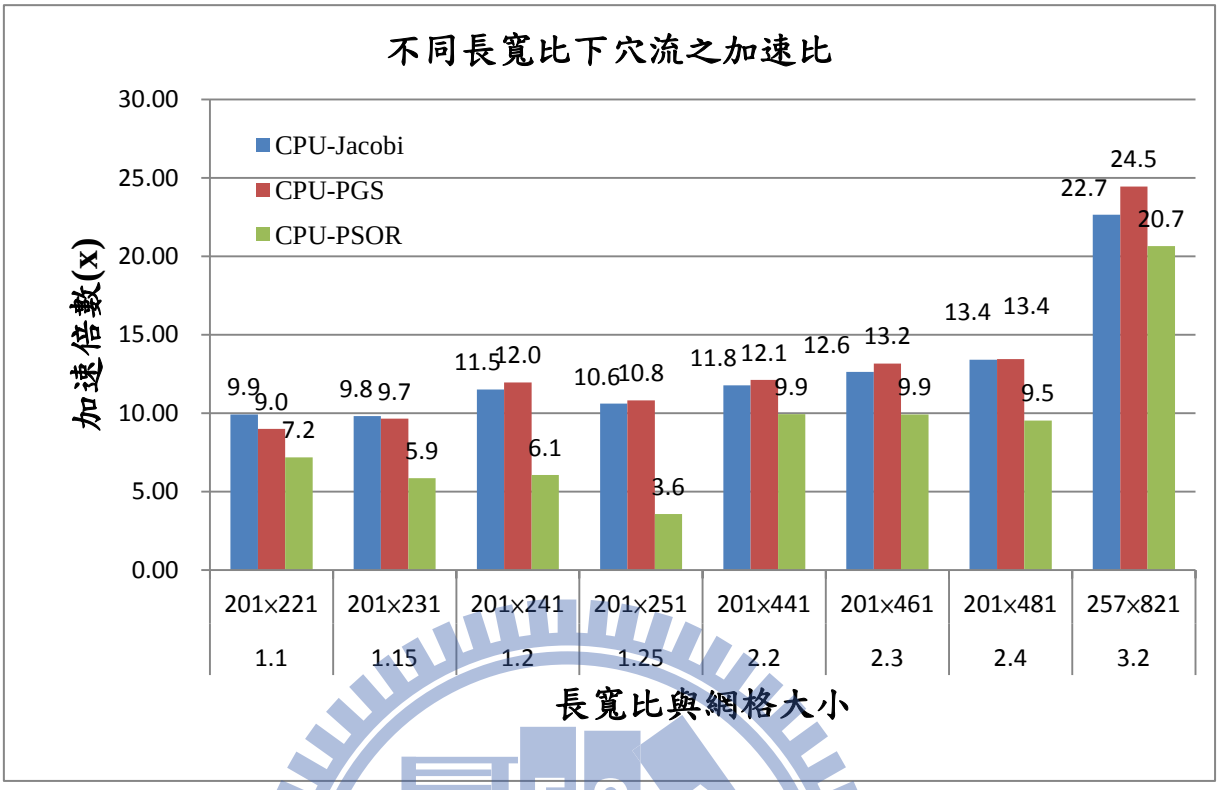


圖 5- 48 不同長寬比下穴流之加速比 2

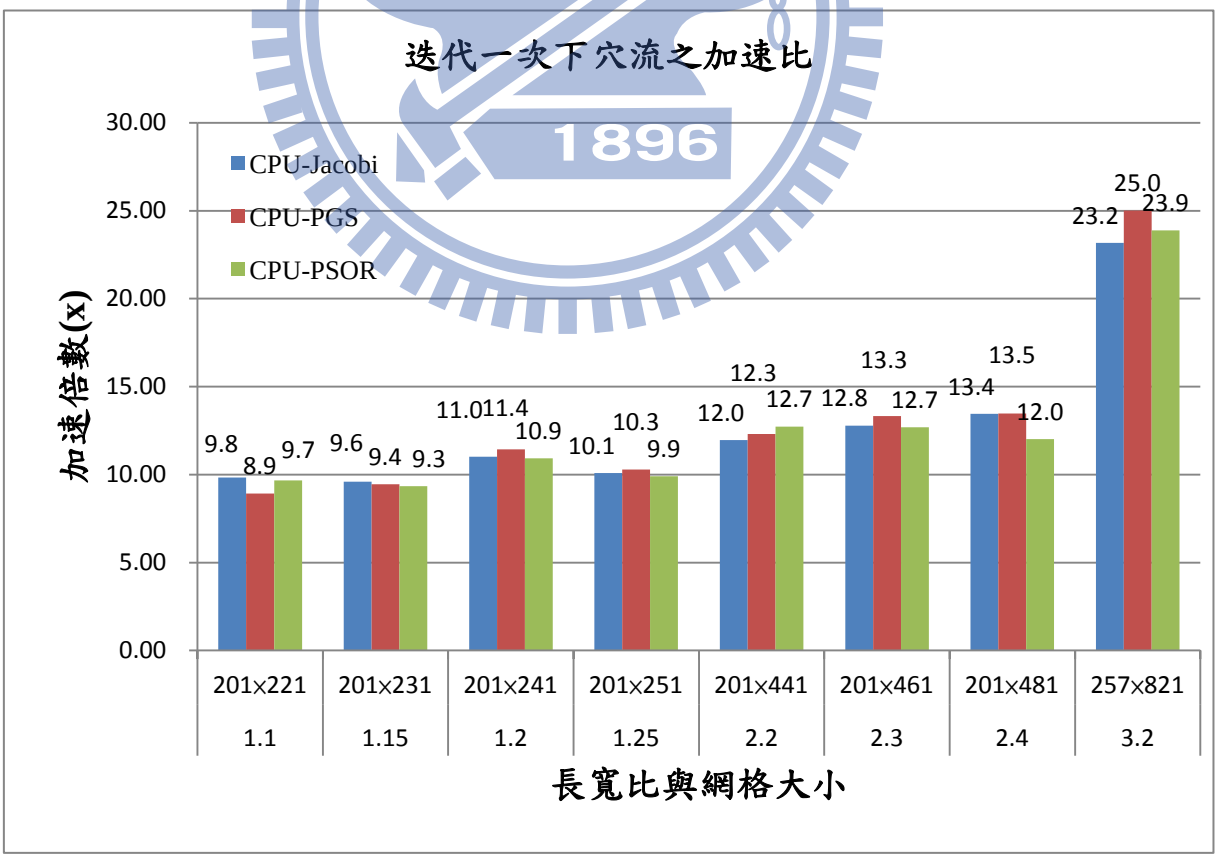


圖 5- 49 迭代一次下穴流模擬加速比

表 5-3 長寬比為 7 之穴流模擬成果表

雷諾數	500	1000	5000
網格大小	257×1793	257×1793	257×1793
GPU-Jacobi			
迭代次數	191063	229633	1410852
模擬時間(S)	331.75	378.407	2877.39
迭代一次時間	0.001736	0.001648	0.002039
CPU-Jacobi			
迭代次數	190147	225266	1406315
模擬時間(S)	11435.78	12611.55	64024.41
總模擬時間加速比	34.47	33.33	22.25
迭代一次時間	0.060142	0.055985	0.045526
迭代一次加速比	34.64	33.97	22.32
CPU-PGS			
迭代次數	190147	225266	1406315
模擬時間(S)	11608.77	13263.2	68695.41
總模擬時間加速比	34.99	35.05	23.87
迭代一次時間	0.061052	0.058878	0.048848
迭代一次加速比	35.16	35.73	23.95
CPU-PSOR			
迭代次數	195152	219104	763575
模擬時間(S)	9117.406	10349.45	36455.67
總模擬時間加速比	27.48	27.35	12.67
迭代一次時間	0.04672	0.047235	0.047743
迭代一次加速比	26.91	28.66	23.41

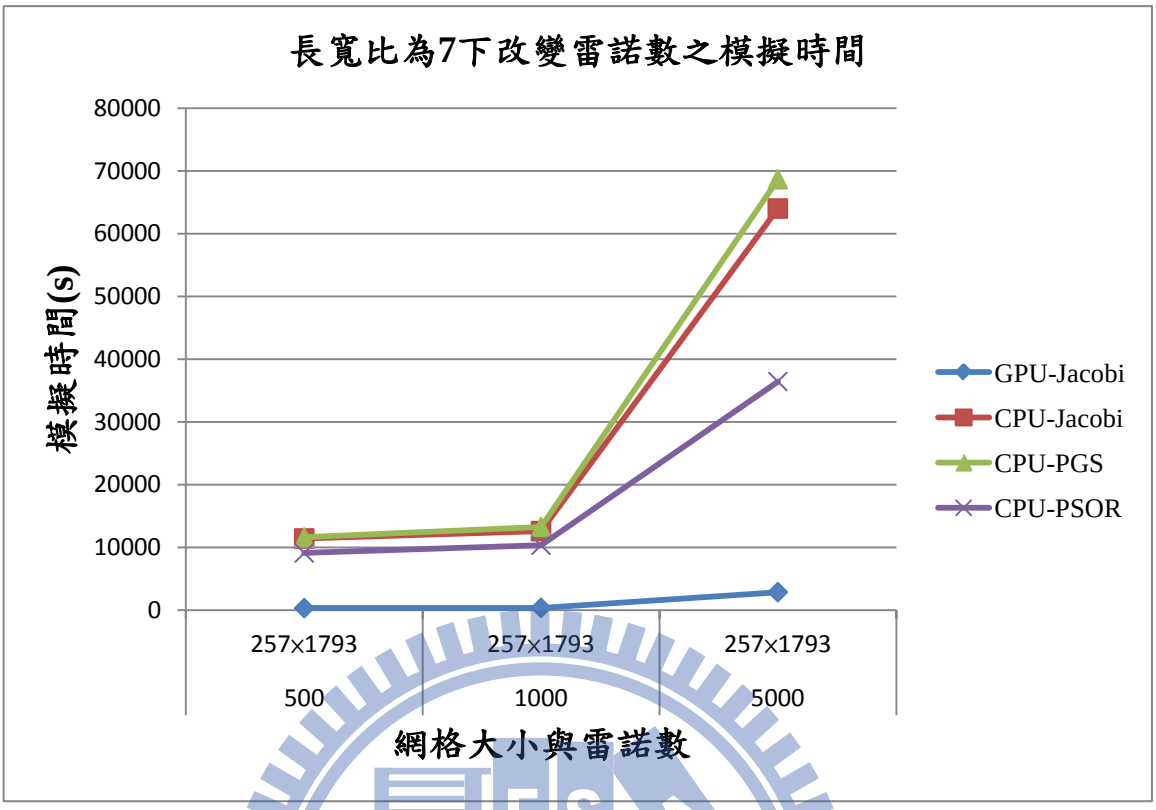


圖 5- 50 長寬比為 7 下不同雷諾數之模擬時間

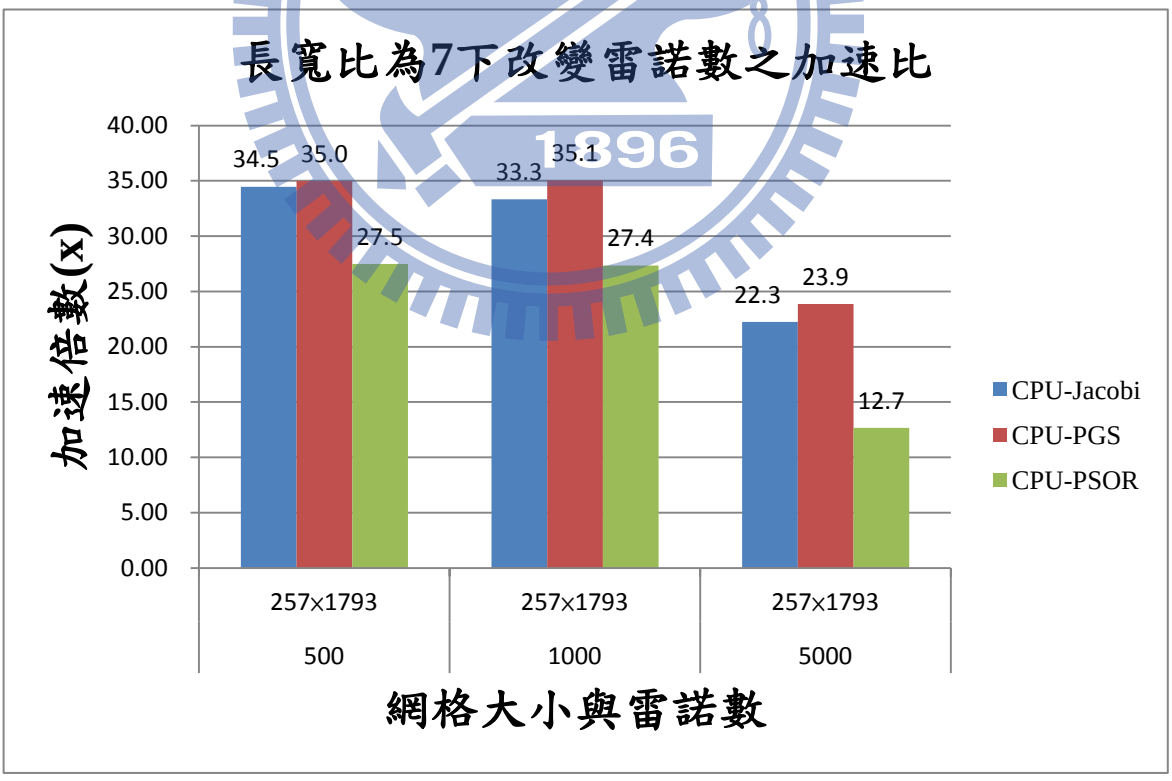


圖 5- 51 長寬比為 7 下不同雷諾數之加速比

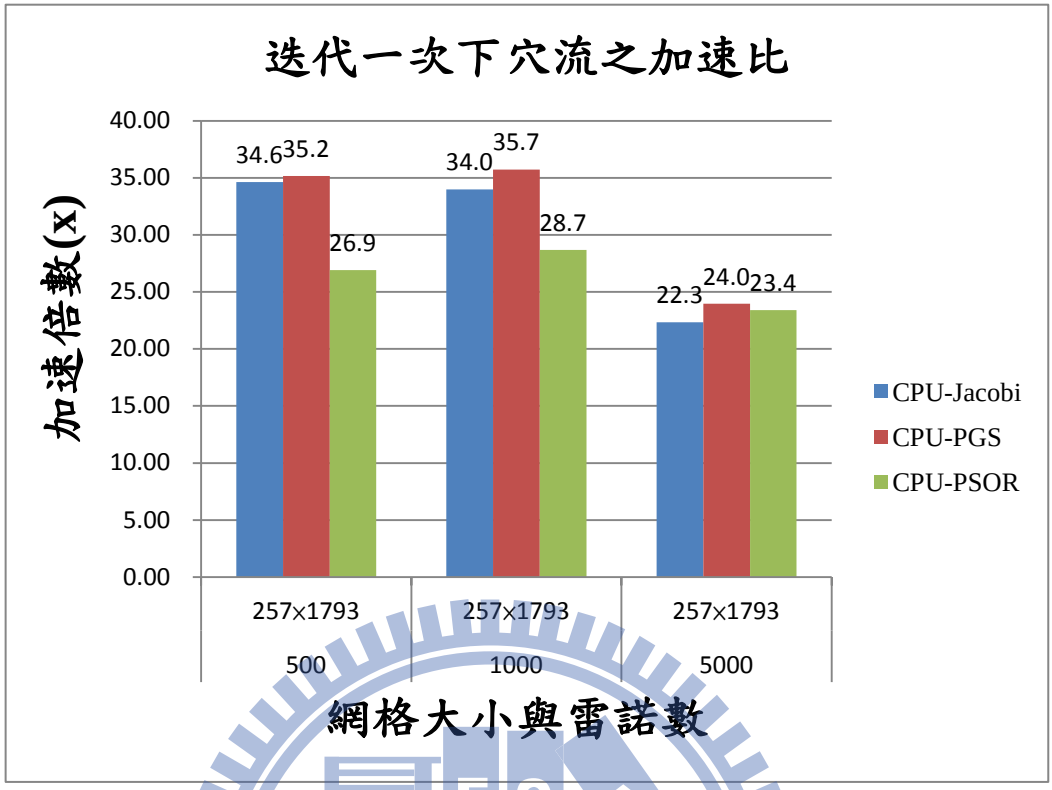


圖 5- 52 迭代一次下穴流模擬加速比

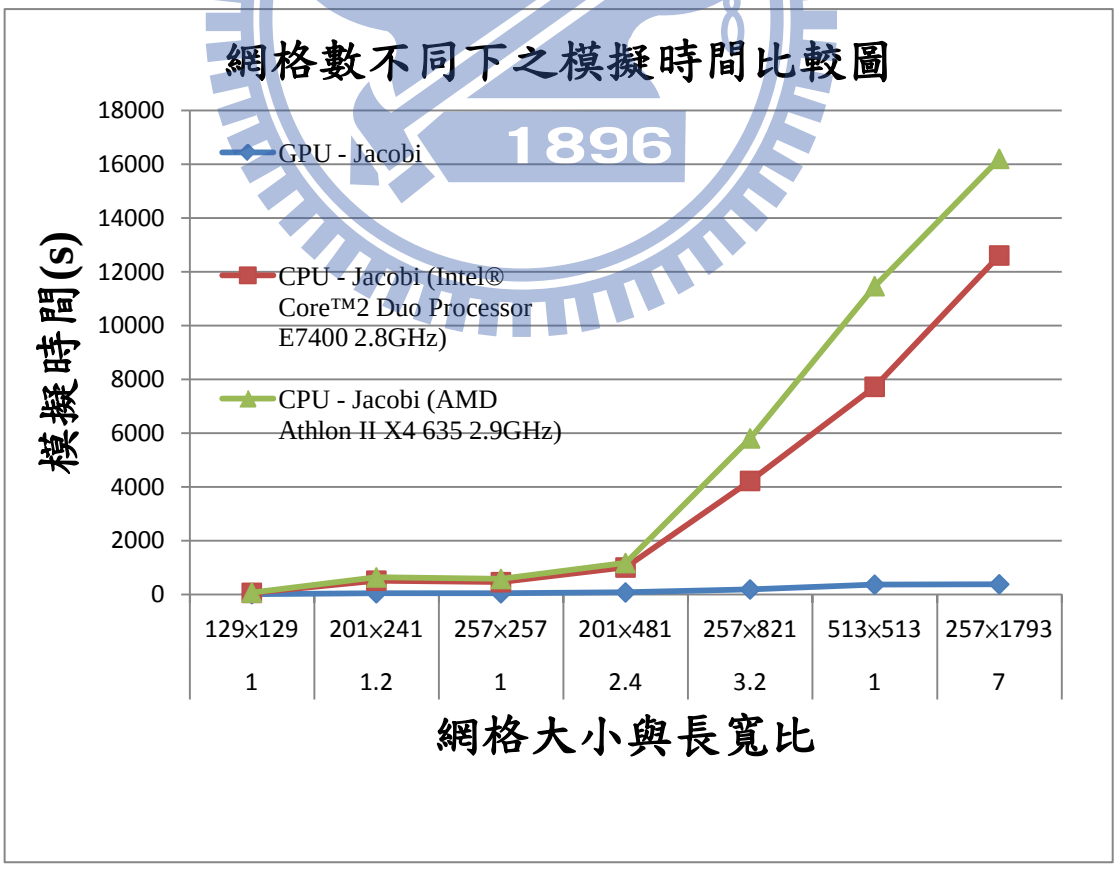


圖 5- 53 網格數不同下之模擬時間圖

表 5-4 不同網格數下之模擬成果表

長寬比 D = H/W	1	1.2	1	2.4	3.2	1	7
網格大小	129×129	201×241	257×257	201×481	257×821	513×513	257×1793
GPU - Jacobi							
迭代次數	102223	185918	140481	183889	230874	352502	229633
模擬時間(S)	13.5	44.469	44	74.75	186.406	363.515	378.407
迭代一次時間	0.000132	0.000239	0.000313	0.000406	0.000807	0.001031	0.001648
CPU - Jacobi (Intel® Core™2 Duo Processor E7400 2.8GHz)							
迭代次數	100190	194435	138896	183482	225736	354359	225266
模擬時間(S)	59.766	512.266	456.516	1002.938	4223.843	7726.063	12611.55
總模擬時間加速比	4.43	11.52	10.38	13.42	22.66	21.25	33.33
迭代一次時間	0.000597	0.002635	0.003287	0.005466	0.018711	0.021803	0.055985
迭代一次加速比	4.52	11.02	10.49	13.45	23.18	21.14	33.97
CPU - Jacobi (AMD Athlon II X4 635 2.9GHz)							
迭代次數	100190	194435	138896	183482	225736	354359	225266
模擬時間(S)	70.484	635.453	577.656	1169.75	5807.156	11470.94	16208.17
總模擬時間加速比	5.22	14.29	13.13	15.65	31.15	31.56	42.83
迭代一次時間	0.000704	0.003268	0.004159	0.006375	0.025725	0.032371	0.071951
迭代一次加速比	5.33	13.66	13.28	15.68	31.86	31.39	43.66

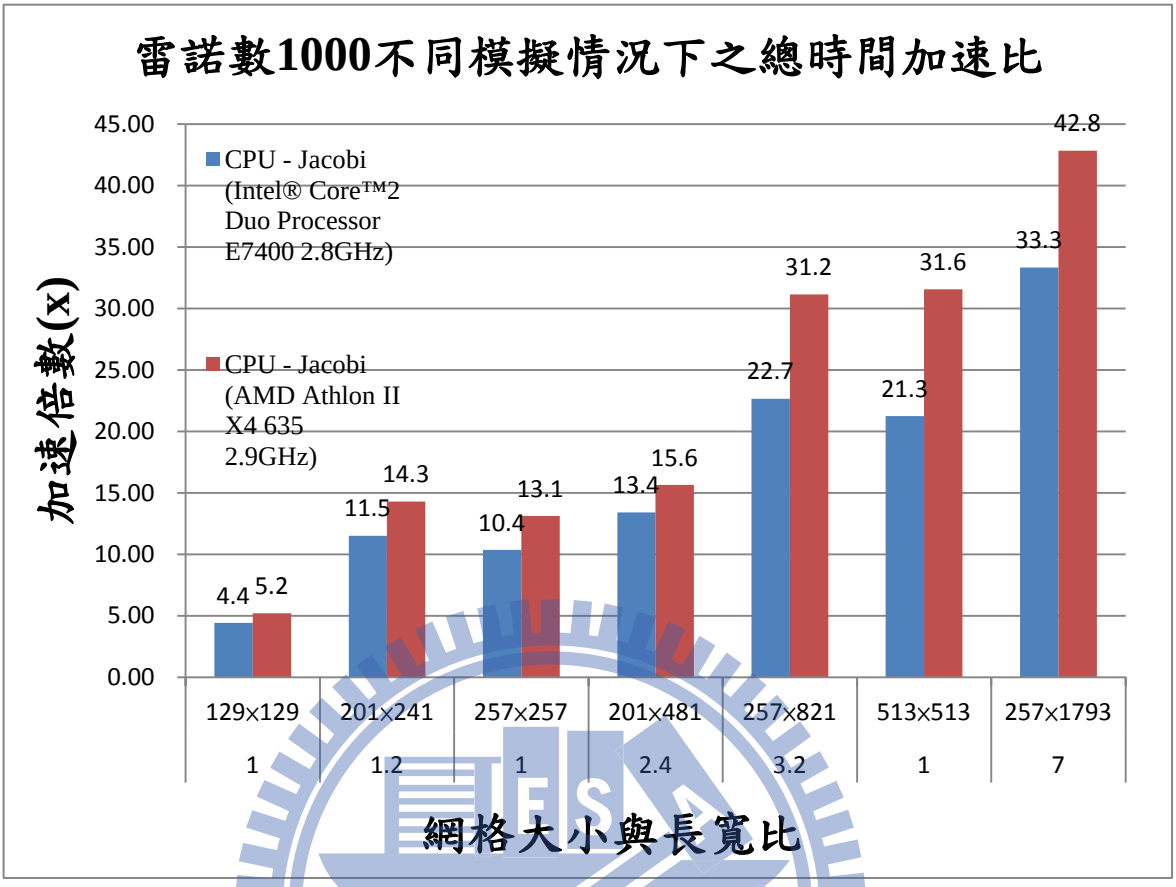


圖 5- 54 雷諾數為 1000 改變網格大小加速比

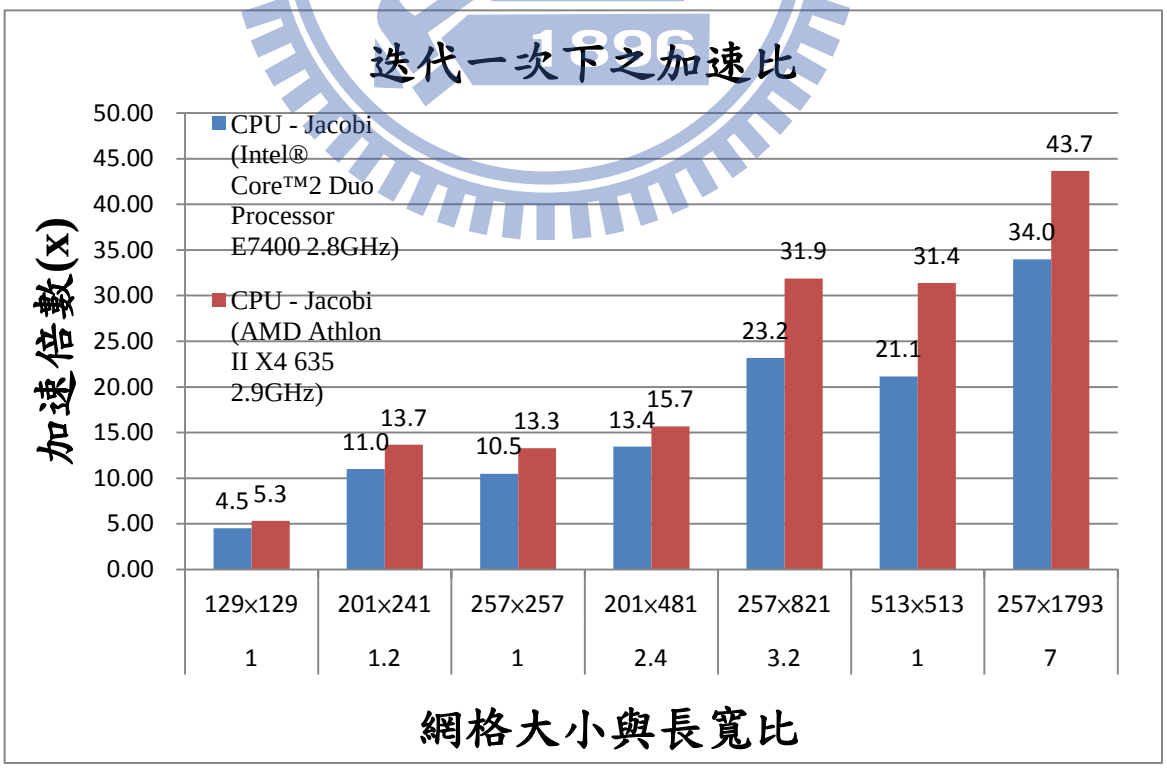
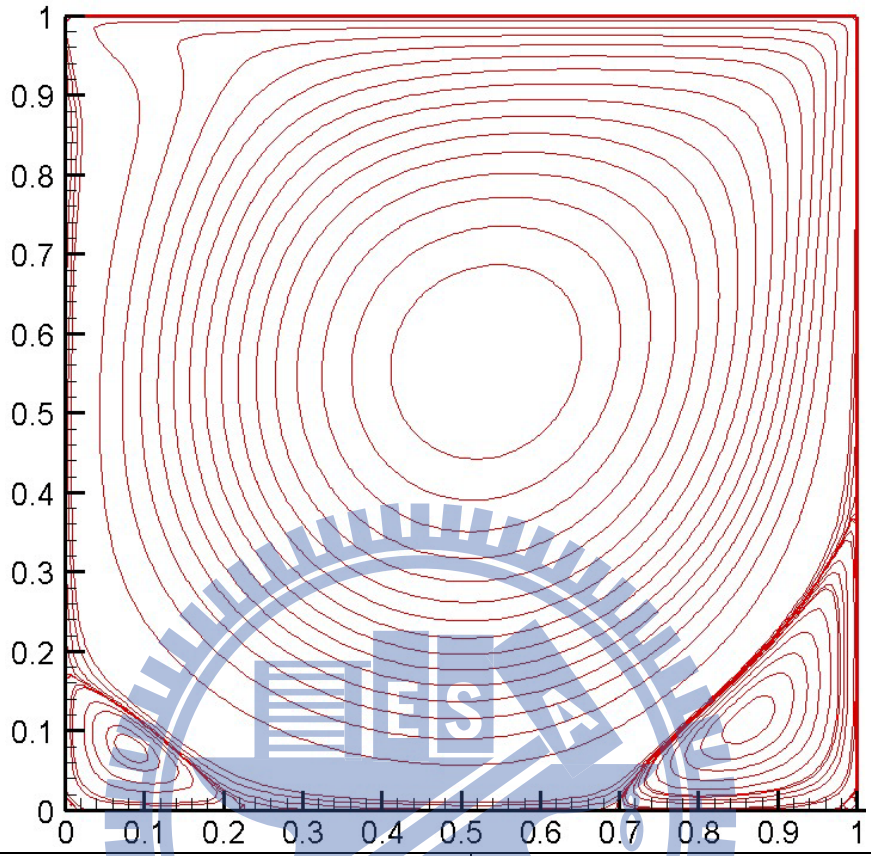
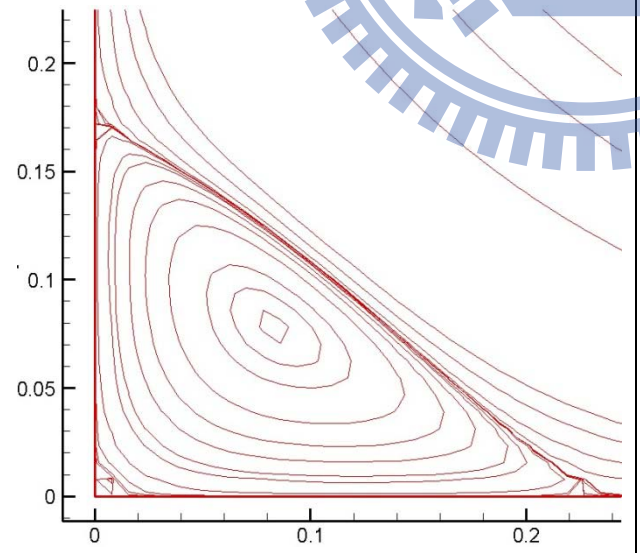


圖 5- 55 不同網格數下之加速比

雷諾數 1000 網格大小 129×129



左下渦旋



右下渦旋

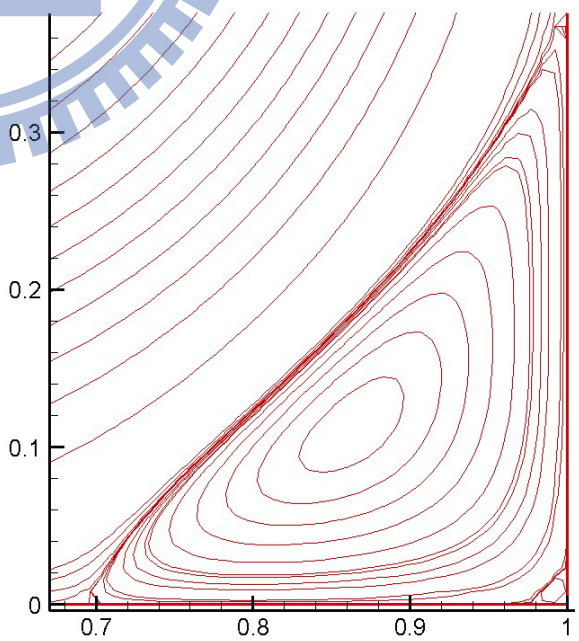


圖 5- 56 雷諾數 1000 網格大小 129×129

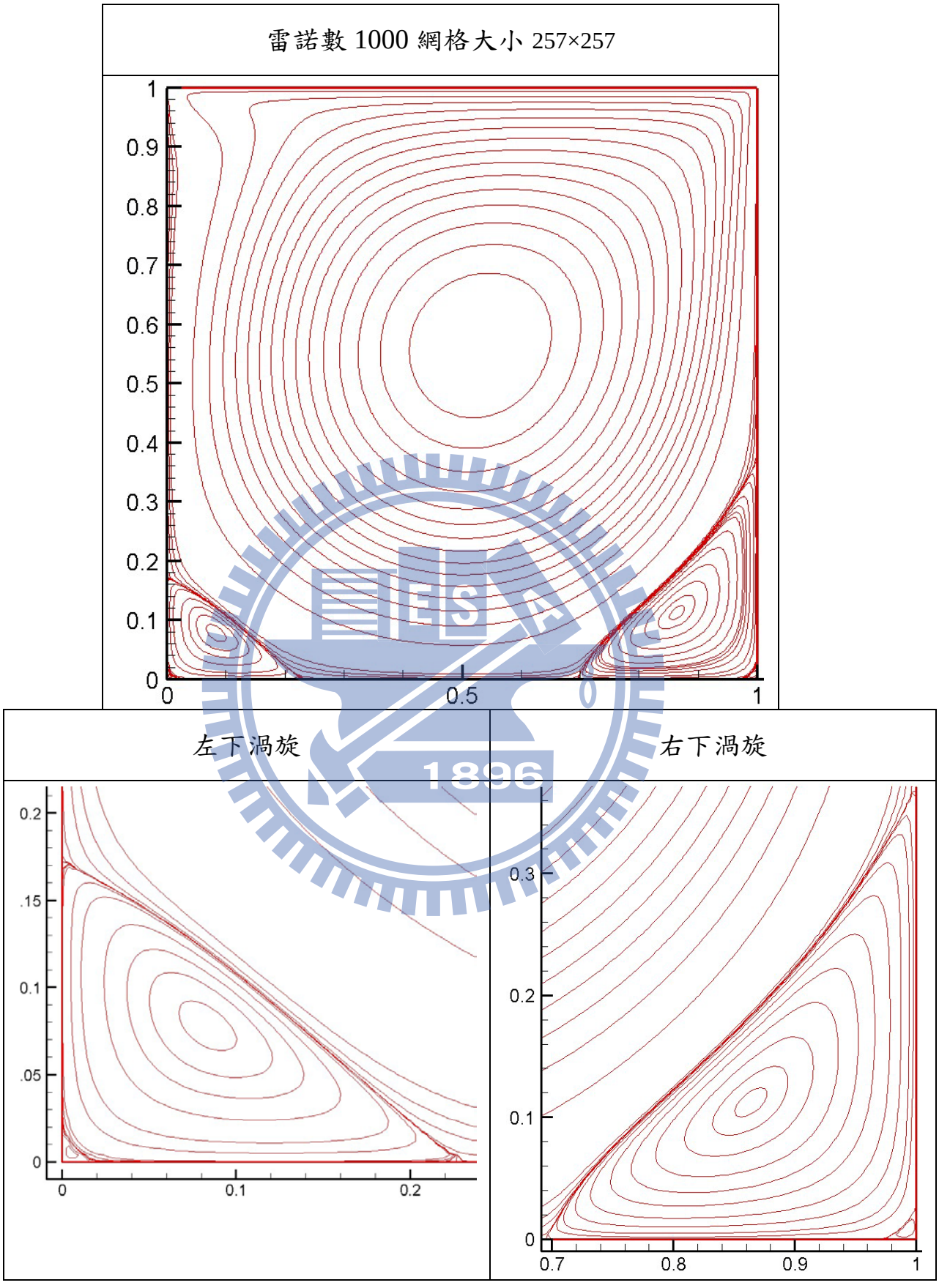
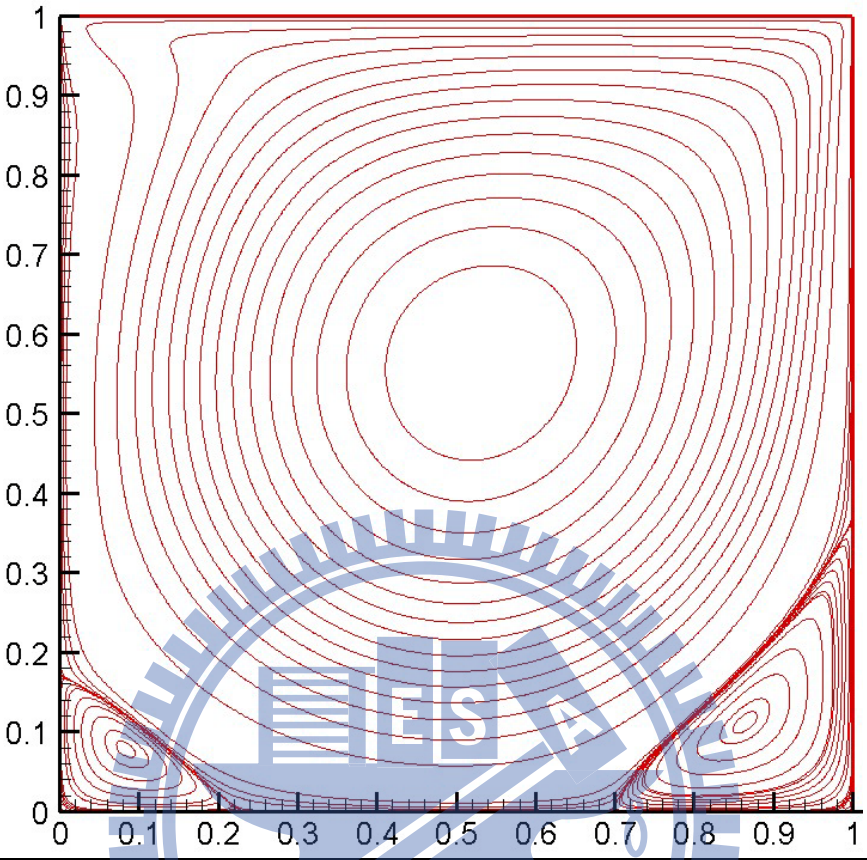
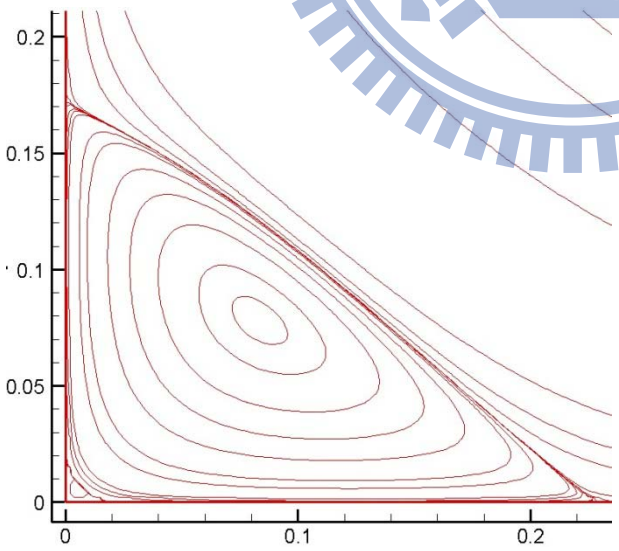


圖 5- 57 雷諾數 1000 網格大小 257×257

雷諾數 1000 網格大小 513×513



左下渦旋



右下渦旋

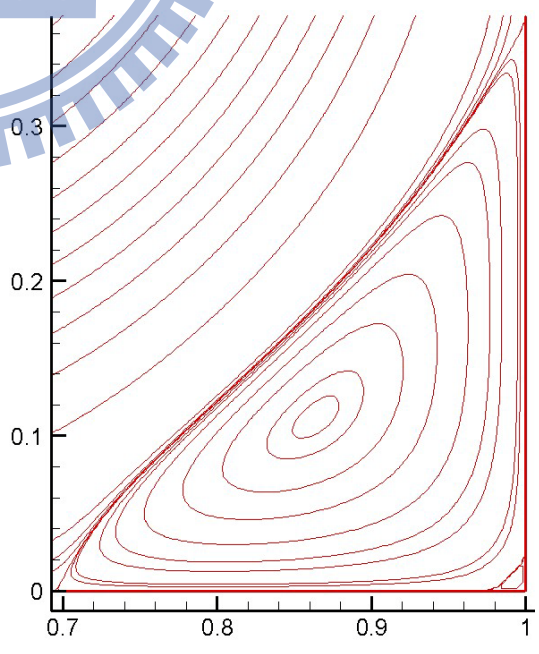


圖 5- 58 雷諾數 1000 網格大小 513×513

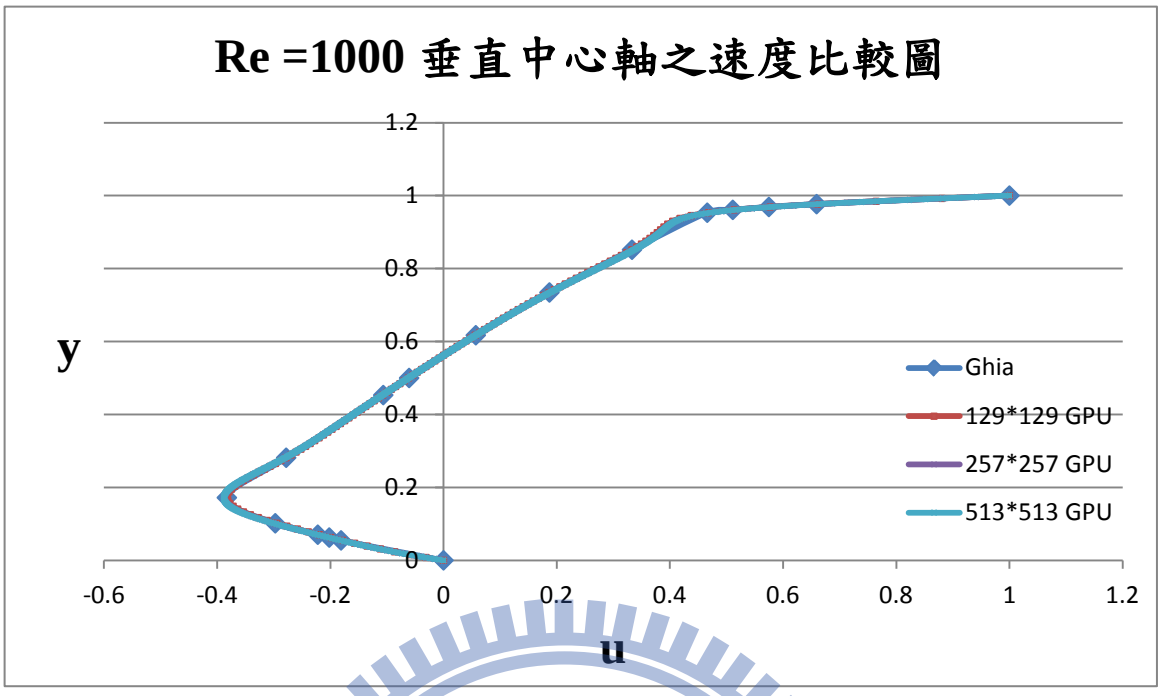


圖 5- 59 Re =1000 垂直中心軸之速度比較圖

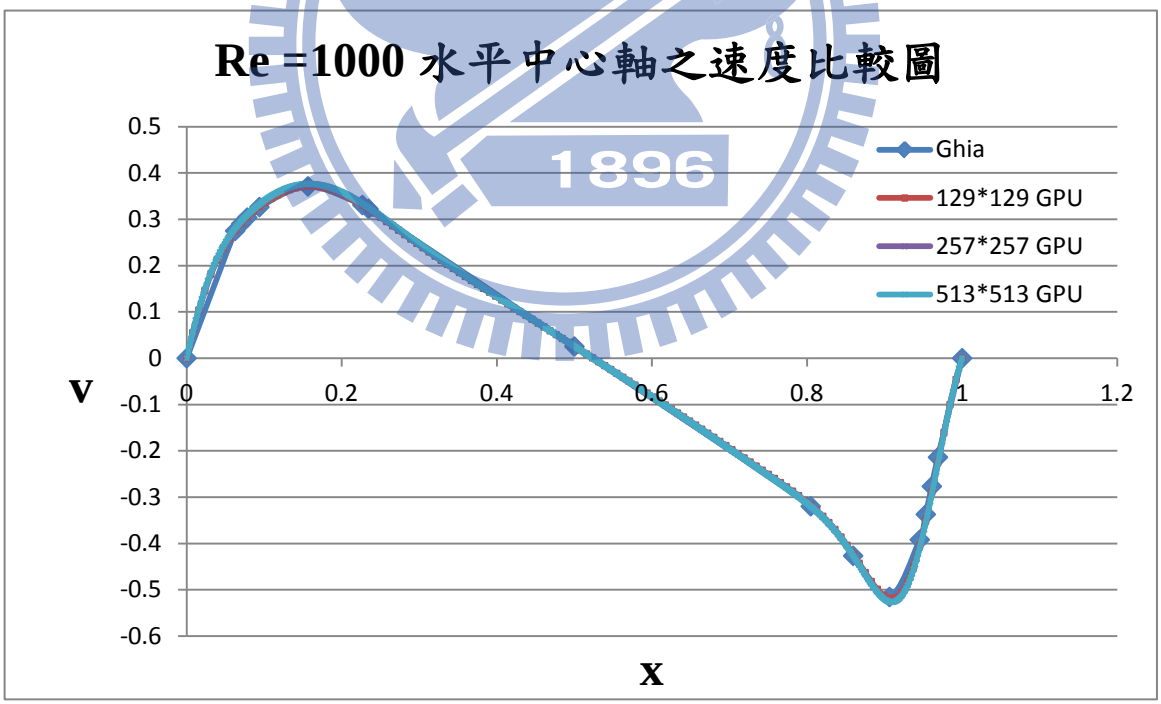


圖 5- 60 Re =1000 水平中心軸之速度比較圖

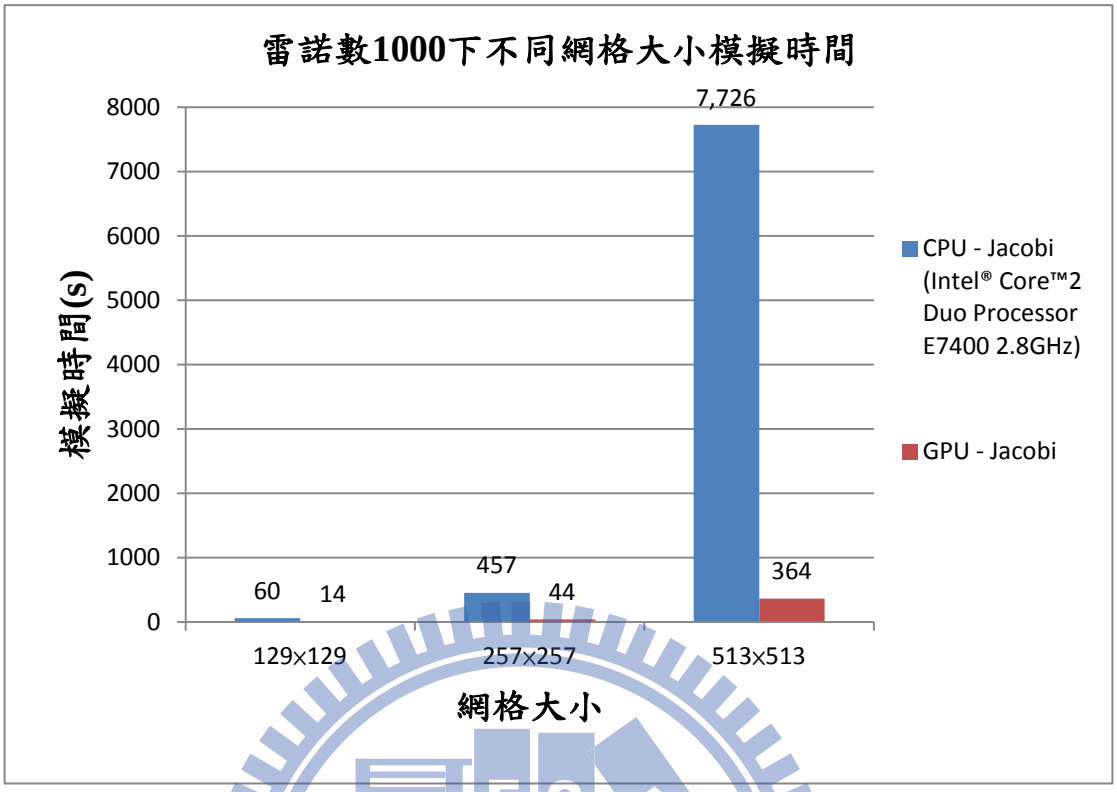


圖 5- 61 雷諾數 1000 下不同網格大小模擬時間

第六章 結論與建議

6.1 結論

本研究藉由二維不可壓縮穴流探討 GPU 加速之成效，透過改變穴流模擬之長寬比與網格大小進行一系列比較。

1. 雷諾數於 1000 時，改變長寬比與網格大小，由圖 5-46 可知網格數越大所得到的加速效益越高，並且在 AMD 與 Intel 兩間公司所開發之中央處理器以 Intel 較適合數值處理，在網格數達 257×1793 下以 AMD 做為基準之加速效果可達 43 倍，Intel 部分可達 33 倍。
2. 網格數量越大下，越有發展平行算法之必要性，由圖 5-55 可發現網格數量為 129×129 下迭代一次所加速之成效略僅有 5 倍而放大網格至 257×257 加速成效達到 11 倍左右，再透過眾多 GPU 計算單元進行平行化演算，其單一次計算量至少達到 65536 時，加速效果較為明顯。
3. 在模擬穴流運動狀況下，邊界部分容易產生震盪，由圖 5-56 至圖 5-58 所示，提高網格數量進行模擬流場，可有效處理邊界之細微現象，但網格加密模擬時間必定拉長，但藉由 GPU 之平行加速計算下，即使在網格加密一倍之下其模擬時間仍較網格略為稀疏之 CPU 運算所耗費之時間來的短，意味著相同模擬時間下，透過平行運算

可以得到更為細緻的結果，如圖 6-1 所示。

4. Fritzsche (2009) 透過 GPU 求解 Navier-Stoke-Equaitons 其針對雷諾數為 1000 下之方穴流進行模擬，本研究進一步調整穴流長寬比及提高雷諾數至 10000 下進行模擬。其模擬時間與加速比如表 6-1。在網格點數達 256×256 下 Fritzsche 之模擬設備為 Intel Quad-core Xeon 2.0GHz 與 nVidia GeForce GTX260 加速成果為 5.8 倍，本研究之加速成效可達 10.4 倍。
5. CUDA 語言至今尚處開發階段，在電腦數值儲存格式上分為單倍精度與雙倍精度，GPU 在雙倍精度格式下尚未發展完善，故其加速效果未達預期，單倍精度格式發展相對較為成熟其加速成效顯著，在單倍精度格式下已能準確求解大部分穴流問題。

6.2 建議

本研究僅以穴流之模擬加以探討 GPU 之效能，建議將來可以進一步探討之方向如下：

1. 本研究求解渦度之數值方法，並非計算效率較高之演算法，可再採用其他數值方法，如多重網格法、共軛梯度法進行求解，可再進一步縮短其模擬時間，但在長型穴流模擬需注意其內部條件設定。
2. 實際穴流流場問題中，不僅是二維空間的現象，若可將其提升至三維空間下模擬，更能顯示其真實流況。

3. 集水區淹水模擬的演算、三維水理演算及三維動床演算，數據規模甚大，可透過 GPU 進行開發，以縮短其龐大模擬時間。

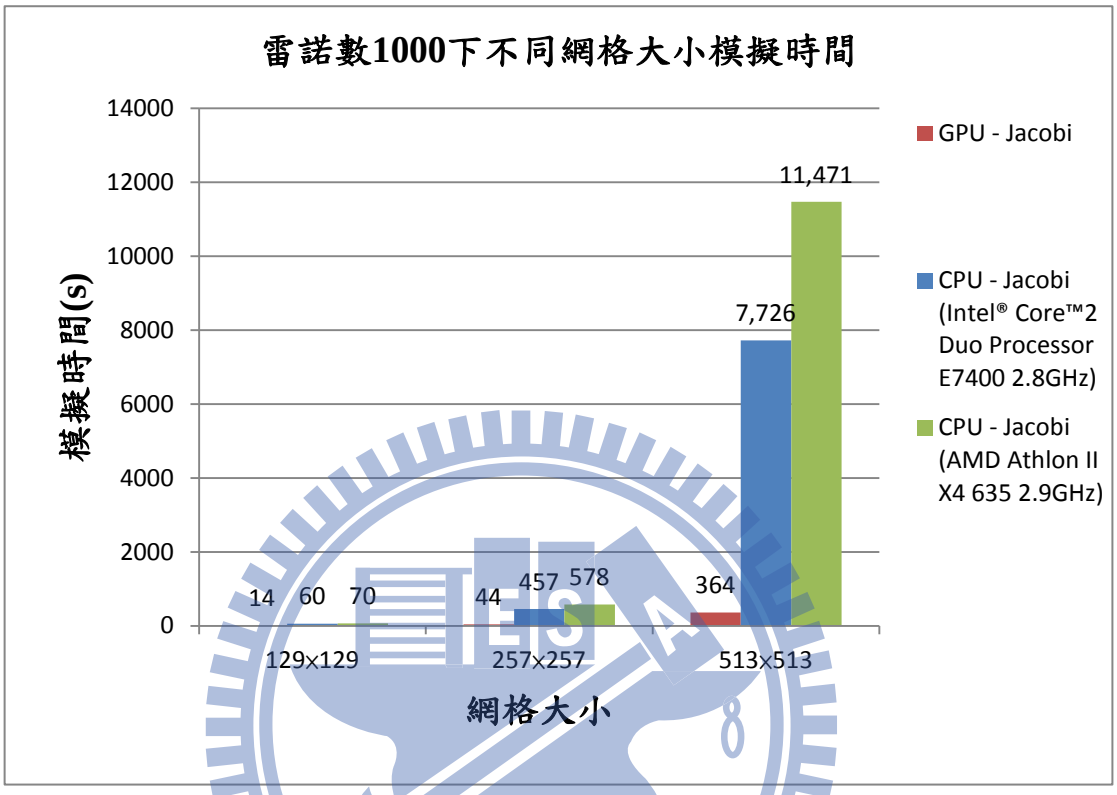


圖 6- 1 雷諾數 1000 下不同網格大小模擬時間

表 6- 1 Fritzsche(2009)與本研究之比較表

			Time in seconds		Total speedup
			Cpu	Gpu	
Fritzsche(2009)	Grid Size	64x64	74	18	4.1
本研究		65x65	5.4	4.7	1.1
Fritzsche(2009)		128x128	905	119	7.6
本研究		129x129	59.77	13.5	4.4
Fritzsche(2009)		256x256	10507	1812	5.8
本研究		257x257	456.52	44	10.4

參考文獻

1. Albensoeder S., Kuhlmann H. C. and Rath H. J., (2001) “Multiplicity of steady two-dimensional flows in two-sided lid-driven cavities.”Theor. Comp. Fluid Dyn.,Vol.14, pp.223-241
2. Benjamin, A. S., and Denny, V. E., (1979) “On the convergence od numerical solution for 2-D flows in a cavity at large Reynolds number”Journal of Computational Physics, Vol.33, pp.340-358
3. Benson, J. D. and Aidun, C. K., (1992) “Transition to unsteady non-periodic states in a through-flow lid-drive cavity” hysics of Fluids A, pp.2316-2319
4. Brandvik, T., Pullan, G., (2008) “Acceleration of a 3D Euler solver using commoditygraphics hardware” in 46th AIAA Aerospace Sciences Meeting and Exhibit
5. Burggraf, O. R., (1966) “Analytical and numerical studies of the structure of steady separated flows” Journal of Fluids Mech. Vol.24, pp.113-151
6. Chen, H. C. and Chen, C. J., (1982) “The finite analytic method”, Inst. of Hydr. Res., Univ. of Iowa, Iowa City, Iowa
7. Chen, H. C. and Chen, C. J., (1984) “Development of finite analytic numerical method for unsteady two-dimensional Navier- Stokes equation” Journal of Computational Physics, Vol.53, pp. 209-226
8. Cheng, M., and Hung, K. C., (2006) “Vortex structure of steady flow in a rectangular cavity” Computers and Fluids , Vol.35, pp.1046-1062
9. Davis, G. D. V. (1983) “Natural convection of air in a square cavity: a bench mark numerical solution” international Journal for Numerical Methods in Fluids, Vol.3, pp. 249-264
10. Egloff, D., (2010) “High Performance Finite Difference PDE Solvers on GPUs” Social Science Research Network, PP. 1-28
11. Freitas, C. J., Street, R.L.,Findikakis, A. N., and Koseff, J. R., (1985) “Numerical simulation of three-dimensional flow in a cavity” International Journal for Numerical Methods in Fluids, Vol.5, pp.561-575
12. Fritzsche, M. (2009) “Parallel numerical simulation of Navier-Stokes-equations on GPUs.” Diplomarbeit, Institut für Angewandte Mathematik, Universität
13. Ghia, U., Chia, K. N., and Shin, C. T., (1982) “High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid Method” Journal of Computational Physics,Vol.48, pp.387-411
14. Goodrich, J. W., Gustafson, K., and Halasi, k., (1990) “Hopf bifurcation in

the deiven cavity.” J.Comput. Phys., Vol.20, pp.219-261
15. HÉRAULT, A., (2010) “SPH on GPU with CUDA” Journal of Hydraulic Research, Vol. 48, Extra Issue, pp. 74–79
16. Kloss, Yu.Yu., Shuvalov,P.V., Tcheremissine, F.G., (2010) “Solving Boltzmann equation on GPU” Procedia Computer Science 1， pp.1083–1091
17. Komatitsch, D., Erlebacher, G., Goddeke, D., Michea, D., (2010) “High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster” Journal of Computational Physics, Vol.229, pp. 7692–7714
18. Koseff, J. R.,and Street, R. L., (1984) “On end wall effects in a lid-driven cavity flow” ASME Journal of Fluids Engineering, Vol.106, pp.385-389
19. Kuhlmann, H. C., Wanschura, M., and Rath, H. J., (1997) “Flow in two-sided lid- driven cavities: Non-uniqueness, instabilities, and cellular structures.”J. Fluid Mech., Vol.336, pp.267-299
20. Kuznik, F., Obrecht, C., Rusaouen, C., Roux, J. J., (2010) “LBM based flow simulation using GPU computing processor” Computers and Mathematics with Applications ,Vol.59 , 2380-2392
21. Liao, S. J., (1992) “Higher-order streamfunction-vorticity formulation of 2D steady-state Navier-Stokes equations” Interational Journal for Numerical Methods in Fluids, Vol.15 pp.595-612
22. Li, M., Tang, T. and Fornberg, B., (1995) “A compact forth-order finite difference scheme for the steady incompressible Navier-Stokes equations.”International Journal for Numerical Methods in Fluids, Vol.20, pp.1137-1151
23. nVIDIA, (2010). NVIDIA CUDA Compute Unified Device Architecture programming GuideVersion 3.0,
24. Pan, F., and Acrivos, A., (1967) “Steady flows in rectangular cavitys” Journal of Fluid Mech., Vol.28, pp.643-655
25. Patil, D. V., Lakshmisha,K. N., Rogg, B,. (2006) “Lattice Boltzmann simulation of lid-driven flow in deep cavities” omputers and Fluids , Vol.35, pp.116-1125
26. Prasad, A. K., and Koseff, J. R.,(1989) “Reynolds number and end-wall effects on a lid-driven cavity flow” Physics of Fluids A, Vol.1, pp.208-218
27. Ramaswamy, B., Tue, T. C., and Akin, T. E., (1992) “Finite-element analysis of unsteady two-dimensional Navier-Stokes equations” Numerical Heat Transfer, Part B, Vol.21, pp.147-182
28. Ramanan, N. and Homsy, G. M., (1994) “Linear stability of lid-driven

cavity flow.” Phys.Fluids, Vol.8, pp.2690-2701
29. Schreibler, R.and Keller, H. B., (1983) “Driven cavity flows by efficient numerical techniques” Journal of computational physics, Vol.49, pp.310-333
30. Shen, J., (1991) “Hopf bifurcation of the unsteady regularized driven cavity flow.” J. Compt. Phys., Vol.95, pp.228-245
31. Spotz, W. F., (1998) “Accuracy and performance of numerical wall boundary conditions for steady, 2D, incompressible streamfuction vorticity” International Journal for Numerical Methods in Fluids, Vol.28, pp.737-757
32. Tölke, J., (2008) “Implementation of a Lattice Boltzmann kernel using the ComputeUnified Device Architecture developed by nVIDIA” Journal Computing and Visualization in Science, Vol.13, Issue 1, pp.29-39
33. Thibault, J., and Senocak, I., (2009) “CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows” 47th AIAA Aerospace Sciences Meeting, Orlando FL, paper no:AIAA-2009-758.
34. Zhang, J., (2003) “Numerical simulation of 2D square driven cavity using fourth-order compact finite difference schemes” Computer and Mathematics with Applications., Vol.45, pp.43-52