

國立臺灣大學工學院工程科學及海洋工程研究所

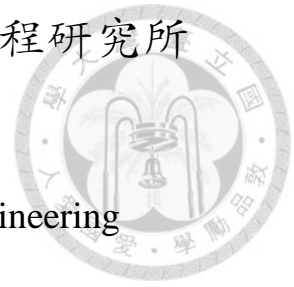
博士論文

Department of Engineering Science & Ocean Engineering

College of Engineering

National Taiwan University

PhD Thesis



發展一波數誤差最佳化有限元素 GPU 平行計算模型

以求解不可壓縮 Navier-Stokes 方程式

On the development of a wavenumber error reduction
finite element model for solving the incompressible
Navier-Stokes equations in parallel on GPU

高仕超

Neo Shih-Chao Kao

指導教授：許文翰 博士

Advisor: Tony Wen-Hann Sheu, Ph.D.

中華民國 104 年 10 月

October, 2015

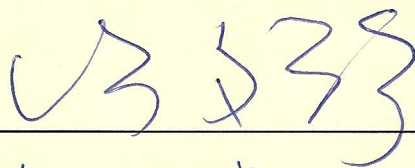
國立臺灣大學博士學位論文
口試委員會審定書

發展一波數誤差最佳化有限元素 GPU 平行計算模型以
求解不可壓縮 Navier-Stokes 方程式

On the development of a wavenumber error reduction finite
element model for solving the incompressible
Navier-Stokes equations in parallel on GPU

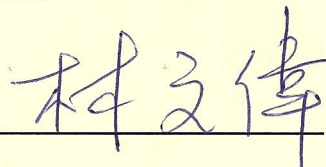
本論文係高仕超君 (D97525011) 在國立臺灣大學工程科學及海
洋工程學系完成之博士學位論文，於民國 104 年 9 月 25 日承下列考
試委員審查通過及口試及格，特此證明

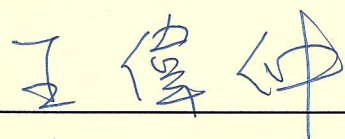
口試委員：

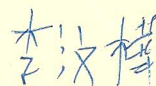


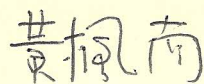
(指導教授)

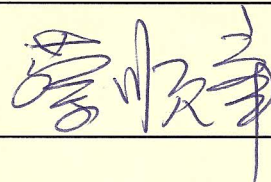






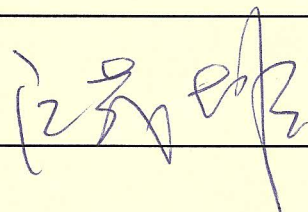








系主任



誌謝

曾經想過，當有一天我可以開始寫誌謝時，開頭應該要如何寫呢？五味雜陳的感覺讓人不知道如何下筆，似乎就是在為自己的最後一里學生生涯進行最後的倒數，每個念頭牽動著過去的回憶，然卻是說不完感謝與幸運。

回想當初進入台大工科系時，心中其實帶有些許不安與徬徨的，自己從理學院數學系轉跨到工學院的工科系，某種程度上來說，我是成一張白紙進來，很多應該早就要知道的知識是沒有的，我曾經懷疑過自己轉來工學院的決定是否正確？而細數這一路下來，我真摯地感謝我的指導教授許文翰博士，憶起當初 2007 年在台北南港所舉辦的中華民國數學年會中只是跟許文翰老師有過一面之緣並短暫交談而已，我在隔年報考博士班並錄取後，去敲許文翰老師辦公室，老師仍然記得我，並且也很快就答應當我的博士論文指導教授，有限元素流體計算是許文翰老師的起家作，也是過往實驗室學長用心維持的學術招牌，許文翰老師願意讓我依著我的研究興趣而從事有限元素流體的計算工作，對一位研究生而言，可以做自己有興趣的研究主題是幸運的！而從許文翰老師身上，我看到的是一位對學術研究投以無盡的熱情及執著，以及對嶄新的研究主題永遠抱以好奇心的學者，這些特質都是我一直希望努力達成的目標，未來期望自己也能夠有著這樣的熱情與執著。

而在就讀博士班期間，感謝許文翰老師願意提供經費讓我能夠出國參加三次的亞洲國際計算流體力學國際會議(ACFD)，參加此類國際會議讓我必須學著用英文發表自己的研究成果，這對於一個博士班學生實屬難得的機會，會議之餘也能夠做個旅遊，其中 5 年前在香港科技大學參加 ACFD8 時，跟著許文翰老師一起走在香港街道上，目的只是為了找一碗好吃的粥，以及去年在南韓濟州島參加 ACFD10 時，跟著許老師及師母共乘一輛計程車遊覽濟州島各地景點，吃著韓國炸雞搭配生啤酒，以及吃韓國烤肉等的美好回憶至今仍然歷歷在目。

除了學術研究以外，我亦耳濡目染許文翰老師的行事風格處事態度，讓我學到很多做人處事的道理，至今仍記得在香港參加 ACFD8 時，某天晚上許文翰老師或許是喝了些酒，而我因為想準備博士候選人資格考而未去參加晚宴，許文翰老師回來後，在會館的客廳對我未去參加晚宴的行為頗有微詞，接著對我訓戒了一番，對當時的我來說，因那時我的父親已經不在人世了，許文翰老師當時的教誨感覺像是父親對兒子的教誨一樣，至今仍難以忘記。而老師對我的寬容也感恩在心頭，當我因一時糊塗而犯了一個大錯時，許文翰老師知道並責備我之餘，仍然還是帶著我一起

以成熟的方式去解決我所犯的錯誤，這讓我深自反省並永銘記於心。

在論文審查期間，感謝台大數學系陳宜良教授、王偉仲教授、交大數學系林文偉教授、成大機械系 Matthew Smith 教授、中央數學系黃楓南教授、海大輪機系蔡順峯教授以及台大工科系蔡武廷教授在論文上的費心審查及提出的寶貴意見，其中特別感謝 Matthew Smith 教授於口試結束後，願意花他寶貴的時間幫我修正論文文法上的錯誤，使得本論文能夠更加完美跟充實。

在就讀博士班過程中，承蒙許文翰老師在計算流體力學知識的教導外、亦承蒙許多學術前輩的教導與指點，感謝台大工科系蔣德普教授在 Fortran 程式設計課程的嚴格訓練，回想當初修課時，每個周末都必須窩在實驗室構思著課程作業的撰寫，雖然過程辛苦但卻很紮實，至今回想起來是受益匪淺；感謝中央數學系的黃楓南教授以及楊肅煜教授，黃教授是我碩士班的指導教授，是我在計算流體力學領域的啟蒙老師；楊教授雖非我指導老師，但每次學生拜訪您時，您都願意花許多時間跟學生對談，讓學生收穫良多；感謝交大機械系吳宗信教授及清大動機系林昭安教授，跟您們兩位前輩在濟州島的星巴克一起邊喝咖啡邊聊天是一個很棒的回憶；感謝義守財務與計算數學系的老師對我以前教導，尤其感謝我以前的導師謝良諭教授，每次回去拜訪您總是一件很快樂的事情。

待在「科學計算與心血管模擬實驗室」的日子，會是很美好的一段回憶，這當中有苦有甜，感謝王明睿(王明達)博士、蔡順峯教授、徐銘辰教授，沒有你們過往在有限元素流體計算的工作，大概不會有這篇論文的誕生。感謝林瑞國教授，你的人生經驗總是惠我良多。感謝目前在新加坡 A*STAR 工作的邱柏雄博士及中國浙江大學物理海洋研究所游景皓教授，過往跟你們的學術討論都讓我受益匪淺。感謝一群跟我一起待過實驗室的學弟妹們：小捲(明宏)、毅瑋、小皓(皓煒)、韋仁、Maya(貞朋)、Yago(鼎盛)、倫語、阿梅(嘉敏)、小白(哲安)、大柱(禹鑫)、日陽、聖宗、育瑋、林樂、豫潔。以及目前在實驗室奮鬥的 Yannick Deleuze(亞霓)博士、Rex(冠碩)、A 胖(承佑)、宇倫、耀宇、千渝跟翁凡，希望你們大家以後要畢業的能順利畢業，已經畢業的都能一切順心。此外，感謝在臺大理論科學研究中心的張覺心博士及 Celine Lo 博士，希望你們接下來的發展也會很順利。

另外感謝我的外國朋友們，感謝目前在苗栗國家衛生研究院的馬克沁(Maxim Solovchuk) 博士，不得不說我的英文聽力會進步真的有很大一部分要歸功於你啊。感謝印度理工學院 Yogesh G. Bhumkar 博士，你是我第一個印度朋友，你的客氣謙遜

跟工作熱忱態度是我想要學習的目標，希望有機會還能在臺灣或印度見面。感謝上海大學王伯福博士，由於年紀相仿，也因此我們彼此很聊得來，也多虧你讓我第一次吃到新疆的皮帶麵（雖然不知道正不正統），希望你在上海大學的教職工作一切順利。感謝北京應用物理與計算數學研究所盛志強博士與中國科學院計算數學與科學工程計算研究所戴小英博士，與你們夫妻相識於法國巴黎，之後再見於中國北京，你們在我待在北京時的盛情款待，讓我至今無法忘懷，也希望你們可以趕快來臺灣玩，好讓我盡盡地主之誼。感謝柳一龍博士，戴著圓框眼鏡的你，看起來就像是一個忠厚老實的大哥哥一樣，希望你接下來待在臺灣的時間可以帶給你美好的回憶。另外我要感謝法國巴黎第六大學 Marc Thiriet 教授，記得跟你及盛志強博士連續兩天爬法國馬賽 Lomeli 大學旁邊的那座山，差點被累死，以及你每次來臺灣訪問時，看到我時的那股爽朗笑聲 ”Hey Mr. Kao!!”，以及過去幾次一起去爬山，這些都是我很好且快樂的回憶。

感謝我最摯愛的家人，感謝我生命中最重要三位女性，我媽媽及我兩個姐姐，感謝妳們長久以來的一直支持、寬容與鼓勵，讓我總算能完成博士學位。也將這份感激之情，獻給我來不及看到的父親。另外我要感謝我的小姨、小姨丈跟他的親友們，在我就讀博士班期間，提供我一個棲身處及無私的幫助，在這邊深深感謝他們。

最後，感謝在我就讀博士班期間，曾經陪在我身邊的那個人，她們在我生命中都曾經扮演著無比重要的角色，雖然我們沒能在陪伴對方到最後，但我想我會記得過去的美好，也希望妳們以後都能過得很幸福快樂。

中文摘要

本篇論文中，我們首先提出一能得到對流項的最小波數誤差，且能增進對流項之穩定性之流線上風有限元素模型。為了驗證論文中所提出的有限元素模型，本論文測試了許多具實解及典型的純量傳輸方程及高雷諾數及高瑞利數的黏性不可壓縮流 Navier-Stokes 方程的測試問題。由結果可知，本文所提出之模型，在所有的測試問題中均能有相當好的精確度及收斂斜率。

在使用迭代法求解經由有限元素方法離散三維空間架構下不可壓縮流 Navier-Stokes 方程後所得到的非對稱非正定矩陣方程時，為了避免 Lanczos 及選主元(pivoting)過程而導致求解發散或者求解不收斂問題，我們提出了避免求解原始矩陣方程，改以求解經由在原始矩陣方程兩側乘上其轉置矩陣所得到的正規矩陣方程的策略。正規矩陣方程為一對稱且正定的型式，因此可利用具求解效率高及無條件收斂特性的共軛梯度(CG)迭代法來求解以得到無條件穩定收斂的答案。本文亦提出了兩種基於多項式型式的預條件子來降低因正規化過程中而大幅提高的條件數以加快收斂速度。由數值測試結果，顯示本文所提出的求解策略效率比起傳統使用 BICGSTAB 及 GMRES 迭代法求解原始矩陣方程還更來得穩定且更有效率。

為了加速計算成本繁重的有限元素方法用以求解不可壓縮流 Navier-Stokes 方程，本文遂將所發展的有限元素計算程式執行在比起中央處理器(CPU)有更高的浮點數運算效能及更大的記憶體帶寬的圖形處理器(GPU)上。此外，本文提出了一些最佳化策略以最佳化其計算效能。對於測試的典型拉穴流問題，本論文所提出的 CPU/GPU 異構平行計算方法的計算加速比及效能比均具相當良好的結果。

最後，本論文將所發展的三維有限元素不可壓縮流體求解器用以求解九十度彎管流及後向階梯流體問題，其計算結果與前人所模擬與實驗之結果均相當的吻合。顯示本文所發展的基於 GPU 的有限元素流體求解器為一精準且可信賴的計算工具。

關鍵字：波數；有限元素；流線上風；CUDA；圖形處理器

Abstract

In this dissertation, a new streamline upwind finite element model which accommodates a minimum wavenumber error for convection terms shown in the transport equation, is presented to enhance convective stability. The validity of the proposed finite element model is justified by solving several problems amenable to analytical and benchmark solutions at high Reynolds and Rayleigh numbers. The results with good accuracy and spatial rate of convergence are demonstrated for all the investigated problems.

To avoid Lanczos or pivoting breakdown while solving the resulting large-scaled un-symmetric and indefinite matrix equations using the mixed finite element formulation, the matrix equations have been modified by pre-multiplying matrix with its transpose counterpart. The resulting normalized matrix system becomes symmetric and positive-definite. A computationally efficient conjugate gradient Krylov iterative solver can be therefore applied to get the unconditionally convergent solution. To improve the slow convergence behavior arising from the increased condition number, the two polynomial-based pre-conditioners are adopted. The numerical results show that the performance of the pre-conditioned conjugate-gradient solver for the normalized system is better than the two common used BICGSTAB and GMRES solvers for the original matrix equations.

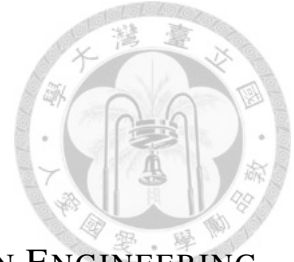
In order to accelerate the time-consuming finite element calculations for the incompressible Navier-Stokes equations, the developed finite element program has been implemented on a hybrid CPU/GPU platform endowed with its high floating-points arithmetic operation performance and large memory bandwidth compared to that implemented in CPU. Moreover, some optimization strategies are introduced in order to optimize the speedup performance. The resulting speedup and efficiency are good for the simulation of benchmark lid-driven cavity flow problem.

Finally, the proposed GPU-based finite element fluid solver was used to investigate the three-dimensional 90 bend curved flow and three-dimensional backward-facing step flow problems. The results simulated from the proposed GPU-based finite element flow solver agree well with other numerical and experimental results. It shows that the proposed GPU-based finite element solver is accurate and reliable for use.

Keywords : wavenumber ; finite element ; streamline upwind ; CUDA ; GPU

NATIONAL TAIWAN UNIVERSITY

DEPARTMENT OF ENGINEERING SCIENCE AND OCEAN ENGINEERING



**On the development of a wavenumber error reducing
finite element model for solving the incompressible
Navier-Stokes equations in parallel on GPU**

By

Neo Shih-Chao Kao

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

AT

NATIONAL TAIWAN UNIVERSITY

TAIPEI, TAIWAN

JULY, 2015

©Copyright by *Neo Shih-Chao Kao* 2015

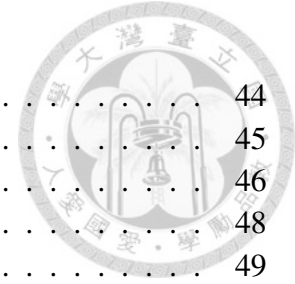


To
My dearest mother
My sisters (Amanda & Samantha)
and
My deceased father

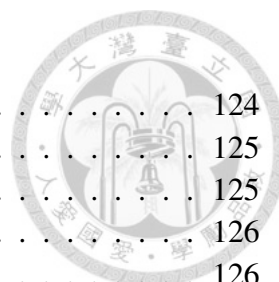


Contents

1	Introduction	1
1.1	Finite element method	3
1.2	Parallel computation	4
1.3	Objectives	6
1.4	Outlines of this study	7
2	Incompressible Navier-Stokes equations	9
2.1	Velocity-pressure variables formulation	9
2.2	Streamfunction-vorticity formulation	11
2.3	Velocity-vorticity formulation	14
2.4	Solution algorithms	15
2.5	Computation challenges	16
3	Finite element model for the transport equation	18
3.1	Introduction	18
3.2	Governing equation	18
3.3	Weak formulation	20
3.4	Finite element model	21
3.4.1	Literature review	21
3.4.2	Streamline Upwind Petrov-Galerkin model	23
3.4.3	Modified wavenumber error optimizing model	24
3.5	Numerical studies	26
3.5.1	Verification study	27
3.5.2	Skew problem	27
3.5.3	Smith & Hutton problem	28
3.5.4	Rotation shaped cone problem	28
3.5.5	Mixing of warm and cold fluids problem	29
4	Finite element model for the incompressible Navier-Stokes equations	41
4.1	Introduction	41
4.2	Weak formulation	42



4.3	Incompressibility constraint condition	44
4.4	Interpolation function	45
4.5	Finite element formulation	46
4.6	Verification study	48
4.6.1	Steady-state analytical verification problem	49
4.6.2	Transient analytical verification problem	49
4.7	Numerical results	50
4.7.1	Lid-driven cavity problem	50
4.7.2	Backward facing step flow problem	50
4.7.3	Natural convection problem	51
5	Iterative solver for three-dimensional Navier-Stokes finite element equations	73
5.1	Introduction	73
5.2	Interpolation functions	74
5.3	The finite element formulation	75
5.4	Newton linearization of the nonlinear convection term	76
5.5	The normalization procedure	78
5.6	The element-by-element technique	79
5.7	Implementation of Dirichlet boundary condition on elementary matrix	81
5.8	Implementation of polynomial-based pre-conditioner	84
5.9	Iterative matrix solver	86
5.9.1	The pre-conditioned CG solver	88
5.9.2	The pre-conditioned BICGSTAB solver	90
5.9.3	The pre-conditioned GMRES solver	92
5.10	Numerical results	94
5.10.1	Verification study	94
5.10.2	Lid-driven cavity flow in a cube	95
5.11	Conclusion remarks	96
6	Parallel computing on GPU	109
6.1	Introduction	110
6.2	GPU architecture	112
6.2.1	Memory hierarchy	114
6.3	CUDA programming model	115
6.4	CUDA Fortran programming	116
6.5	Implementation of iterative solvers on GPU	119
6.5.1	Inner product operation on GPU	120
6.5.2	Matrix-vector product operation on GPU	121
6.6	Optimization	123
6.6.1	Global memory coalescing	123

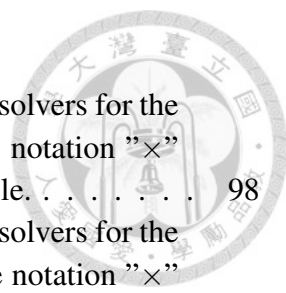


6.6.2	Shared memory	124
6.7	Description of GPU code	125
6.8	Numerical study	125
6.8.1	2D verification study	126
6.8.2	3D verification study	126
6.8.3	Lid-driven cavity flow problem	127
6.9	Performance analysis	127
6.9.1	Introduction	127
7	Applications	146
7.1	Three-dimensional 90 bend curved duct flow problem	146
7.1.1	Problem description and validation	147
7.1.2	Numerical results	148
7.2	Three-dimensional backward facing step flow problem	149
7.2.1	Problem description and numerical results	150
8	Concluding Remarks	163
	Bibliography	165



List of Tables

3.1	The computed L_2 -error norms and the corresponding spatial rates of convergence (R.O.C.) for the calculations carried out at four chosen meshes using the proposed SUPG-FEM and MWE-FEM model for $Re = 100$. . .	30
3.2	The computed L_2 -error norms and the corresponding spatial rates of convergence (R.O.C.) for the calculations carried out at four chosen meshes using the proposed SUPG-FEM and MWE-FEM model for $Re = 1000$. . .	30
4.1	The predicted L_2 error norms for the steady-state analytical verification problem considered in Sec. 4.6.1.	53
4.2	The predicted L_2 error norms at $t = 4$ for the transient analytical verification problem considered in Sec. 4.6.2.	54
4.3	The predicted L_2 error norms at $t = 8$ for the transient analytical verification problem considered in Sec. 4.6.2.	54
4.4	Comparison of the predicted streamfunction solutions with the results of Ghia [52] for the lid-driven cavity problem considered in Sec. 4.7.1. . . .	55
4.5	Comparison of the predicted streamfunction solutions with the results of Erturk [53] for the lid-driven cavity problem considered in Sec. 4.7.1. . . .	56
4.6	Comparison of the reattachment and separation lengths for the backward-facing step flow problem investigated at $Re = 800$	57
4.7	Comparison of the predicted streamfunction at $(0.5,0.5)$ for the natural convection problem considered in Sec. 4.7.3.	58
4.8	Comparison of the predicted averaged Nusselt numbers \overline{Nu} for the natural convection problem considered in Sec. 4.7.3.	58
5.1	The computed L_2 error norms for the problem considered in Sec. 5.10.1. In this table, ”-” means that the solution is not computable.	97
5.2	The total CPU times for three solvers in 16^3 tri-quadratic elements. . . .	97
5.3	The finite element solutions computed from three iterative solvers for the problem considered in a domain of 21^3 nodal points. The notation ” \times ” shown in this table means that the solution is not computable.	98



5.4 The finite element solutions computed from three iterative solvers for the problem considered in a domain of 41^3 nodal points. The notation "×" shown in this table means that the solution is not computable. 98

5.5 The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 61^3 nodal points. The notation "×" shown in this table means that the solution is not computable. 99

5.6 The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 21^3 nodal points. The notation "×" shown in this table means that the solution is not computable. PJCG : CG iterative solver used together with Jacobi pre-conditioner ; PPCG : CG iterative solver used together with polynomial pre-conditioner. 99

5.7 The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 41^3 nodal points. The notation "×" shown in this table means that the solution is not computable. 100

5.8 The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 61^3 nodal points. The notation "×" shown in this table means that the solution is not computable. 100

6.1 Some key specifications of the considered CPU processor, and the NVIDIA K20 and K40 GPU cards 130

6.2 Some features of different GPU device memory types 131

6.3 The computed L_2 error norms obtained at different grids for the 2D verification problem considered in Sec. 6.8.1. 131

6.4 The computed L_2 error norms obtained at different grids for the 3D verification problem considered in Sec. 6.8.2. 131

6.5 Summary of the most important routines on the implementation platforms 132

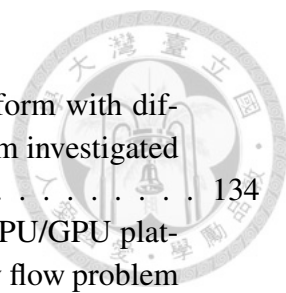
6.6 Meshes used in the performance analysis of the 2D lid-driven cavity problem considered in Sec. 6.8.3. 132

6.7 Meshes used in the performance analysis of the 3D lid-driven cavity problem considered in Sec. 6.8.3. 132

6.8 Timing in seconds of each routine executed on CPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 1000$. The value is "()" denotes the relative time. 133

6.9 Timing in seconds of each routine executed on CPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 5000$. The value is "()" denotes the relative time. 133

6.10 Timing in seconds of each routine executed on CPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 400$. The value is "()" denotes the relative time. 134



6.11 Timing in seconds of each routine executed on CPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 1000$. The value is "()" denotes the relative time. 134

6.12 Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 1000$ 135

6.13 Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 5000$ 135

6.14 Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 400$ 136

6.15 Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 1000$ 136

6.16 Comparisons of the total computation time(s) and the speedup for the 2D lid-driven cavity flow problem investigated at $Re = 1000$ 137

6.17 Comparisons of the total computation time(s) and the speedup for the 2D lid-driven cavity flow problem investigated at $Re = 5000$ 137

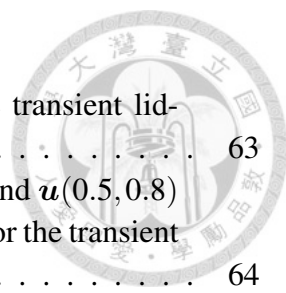
6.18 Comparisons of the total computation time(s) and the speedup for the 3D lid-driven cavity flow problem investigated at $Re = 400$ 137

6.19 Comparisons of the total computation time(s) and the speedup for the 3D lid-driven cavity flow problem investigated at $Re = 1000$ 137



List of Figures

3.1	One-dimensional weighting functions plots for the different finite element model. (a) Galerkin model ; (b) SUPG model ; (c) MWE model . . .	31
3.2	Schematic of the skew problem considered in Sec. 3.5.2	32
3.3	The predicted two- and three-dimensional contours of ϕ for the skew problem considered in Sec. 3.5.2. (a)-(b) $Re = 100$; (c)-(d) $Re = 1000$. . .	33
3.4	Schematic of the Smith & Hutton problem considered in Sec. 3.5.3. . . .	34
3.5	The predicted two- and three-dimensional contours of ϕ for the Smith & Hutton problem considered in Sec. 3.5.3. (a)-(b) $Re = 10^8$	34
3.6	The predicted two- and three-dimensional contours of ϕ for the Smith & Hutton problem considered in Sec. 3.5.3. (a)-(b) $Re=10^8$; (c)-(d) $Re=10^{12}$. . .	35
3.7	(a) Schematic of the rotation shaped cone problem considered in Sec. 3.5.4 ; (b) Initial condition.	36
3.8	The predicted solution contours of ϕ for the rotation shaped cone problem considered in Sec. 3.5.4. (a) $t = \frac{\pi}{2}$; (b) $t = \pi$; (c) $t = \frac{3\pi}{2}$; (d) $t = 2\pi$	37
3.9	(a) Schematic of the mixing warm and cold fluids problem considered in Sec. 3.5.5 ; (b) Initial condition.	38
3.10	The predicted time-evolving temperature contours ϕ for the mixing warm and cold fluids problem considered in Sec. 3.5.5. (a) $t = 1s$; (b) $t = 2s$; (c) $t = 3s$; (d) $t = 4s$	39
3.11	Comparison of predicted solutions with the analytical solutions for the mixing warm and cold fluids problem considered in Sec. 3.5.5, where θ is defined in Fig. 3.9 . (a) $\theta = 0^\circ$; (b) $\theta = 45^\circ$; (c) $\theta = 90^\circ$; (d) $\theta = 135^\circ$	40
4.1	Schematic of the lid-driven cavity problem considered in Sec. 4.7.1 . . .	59
4.2	Residual reduction plots for the steady-state lid-driven cavity problem considered in Sec. 4.7.1. (a)-(b) $Re = 5000$; (c)-(d) $Re = 10000$	60
4.3	Comparison of the streamfunction contours and velocity profiles $u(0.5, y)$ and $v(x, 0.5)$ for the steady-state lid-driven cavity problem considered in Sec. 4.7.1. (a)-(b) $Re = 5000$; (c)-(d) $Re = 10000$	61
4.4	Comparisons of the predicted velocity profiles $u(0.1, y)$, $u(0.9, y)$, $v(x, 0.1)$ and $v(x, 0.9)$ for the steady-state lid-driven cavity problem considered in Sec. 4.7.1. (a)-(b) $Re = 5000$; (c)-(d) $Re = 10000$	62



4.5 Residual reduction plot investigated at $Re = 400$ for the transient lid-driven cavity flow problem considered in Sec. 4.7.1. 63

4.6 Comparison of the predicted velocity profiles $u(0.5, 0.2)$ and $u(0.5, 0.8)$ investigated at $Re = 400$ with the results of Pontaza's [55] for the transient lid-driven cavity problem considered in Sec. 4.7.1. 64

4.7 Comparison of the predicted velocity profile $u(0.5, 0.5)$ investigated at $Re = 400$ with the results of Dailey's [56] for the transient lid-driven cavity considered in Sec. 4.7.1. 64

4.8 Schematic of the backward-facing step flow problem considered in Sec. 4.7.2 (a) Problem geometry and recirculation regions ; (b) Uniform inlet flow profile with an inlet flow channel ; (c) Fully-developed inlet flow profile without an inlet flow channel. 65

4.9 Residual reduction plots for the steady-state backward-facing step flow problem at $Re=800$ considered in Sec. 4.7.2 (a) With an inlet channel ; (b) Without an inlet channel 66

4.10 Comparison of the predicted u -velocity profiles investigated at $Re = 800$ with the results of Gartling [58] and Erturk [59] for the steady-state backward-facing step flow problem considered in Sec. 4.7.2 (a) $x = 3$; (b) $x = 7$ 67

4.11 Comparison of the reattachment lengths x_1/h with the results of Erturk [59] for the backward-facing step flow problem considered in section 4.7.2. 67

4.12 Schematic of the natural convection problem considered in Sec. 4.7.3. . . 68

4.13 Residual reduction plots for the natural convection problem considered in Sec. 4.7.3 (a) $Ra = 10^7$; (b) $Ra = 10^8$ 69

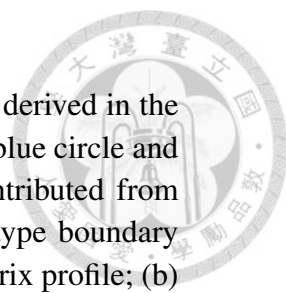
4.14 The streamfunction and T -temperature contours for the natural convection problem considered in Sec. 4.7.3. (a)-(b) $Ra = 10^7$; (c)-(d) $Ra = 10^8$. 70

4.15 Comparisons of the v -velocity profiles with the results of Najafi [67] for the natural convection problem considered in Sec. 4.7.3. (a) $Ra = 10^5 \sim 10^7$; (b) $Ra = 10^8$ 71

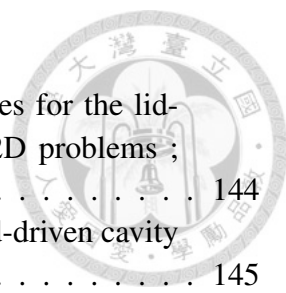
4.16 Comparison of the T -temperature profiles for $10^5 \leq Ra \leq 10^8$ with the results of Najafi [67] for the natural convection problem considered in Sec. 4.7.3. 72

4.17 Comparison of the local Nusselt number distribution at the hot wall for $10^3 \leq Ra \leq 10^7$ with the results of Wan [71] for the natural convection problem considered in Sec. 4.7.3. 72

5.1 Schematic of the primitive variable storing in a tri-quadratic element . . . 101



5.2	Illustration of a 402×402 finite element matrix equation derived in the 2^3 tri-quadratic elements. The green square, red diamond, blue circle and black triangle symbols represent the non-zero entries contributed from the continuity equation, momentum equations, Dirichlet-type boundary data and Newton linearization, respectively. (a) Global matrix profile; (b) Matrix profile in the dashed block of matrix (a).	102
5.3	Illustration of the Dirichlet boundary condition implementation on boundary elementary matrix	103
5.4	Computed eigenvalues for the matrix equation $\underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{D}}}^{-1} \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}}$	104
5.5	Predicted spectral radius of $\underline{\underline{\mathbf{I}}} - \omega \underline{\underline{\mathbf{D}}}^{-1} \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}}$ versus the scaling parameter ω	104
5.6	Computed eigenvalues for the matrix equation $\underline{\underline{\mathbf{I}}} - \omega \underline{\underline{\mathbf{D}}}^{-1} \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}}$	105
5.7	Predicted spectral radius of $\underline{\underline{\mathbf{I}}} - \omega \underline{\underline{\mathbf{D}}}^{-1} \underline{\underline{\mathbf{A}}}$ versus the scaling parameter ω	105
5.8	Schematic of the three-dimensional lid-driven cavity problem considered in Sec. 5.10.2	106
5.9	Comparison of the predicted velocity profiles at the mid-plane $y = 0.5$. (a) $Re = 400$; (b) $Re = 1000$	106
5.10	The residual reduction plots in one inner iteration (at the fourth outer iteration) using the investigated iterative solvers to solve the incompressible Navier-Stokes equations at $Re = 400$ in 41^3 nodal points. (a) Non pre-conditioned solvers; (b) Pre-conditioned solvers.	107
5.11	The residual reduction plots in one inner iteration (at the fifth outer iteration) using the investigated pre-conditioned iterative solvers to solve the Navier-Stokes equations at $Re = 1000$ in 41^3 nodal points.	108
6.1	Comparison of the peak performance of the CPUs and the GPUs	138
6.2	Comparison of the memory bandwidth of the CPUs and the GPUs	138
6.3	Basic structure of the two investigated typical processors [94]: (a) CPU ; (b) GPU	139
6.4	Basic structure of the Kepler K20 architecture.	139
6.5	Schematic of the CUDA programming model.	140
6.6	Illustration of the 2D mesh coloring strategy in four color-element groups without sharing the same global node.	141
6.7	Illustration of the 3D mesh coloring strategy in eight color-element groups without sharing the same global node.	141
6.8	Distribution of all element matrices in global memory to satisfy the global memory coalescing.	142
6.9	Extension of element matrix to satisfy the global memory coalescing condition.	142
6.10	Illustration of the inner product operation on GPU	143
6.11	Flow chart for the developed code executed on a hybrid CPU/GPU platform.	143



6.12 Comparison of the predicted mid-sectional velocity profiles for the lid-driven cavity problem considered in Sec. 6.8.3. (a)-(b) 2D problems ; (c)-(d) 3D problems. 144

6.13 Plot of speedup ratio for the two- and three-dimensional lid-driven cavity problem at different grids size. 145

7.1 Schematic of the 90 bend duct flow problem considered in Sec. 7.1 152

7.2 Residual reduction plot for the 90-degree bend duct flow problem considered in Sec. 7.1. (grids : 410375) 153

7.3 Grid independence test for the 90-degree bend duct flow problem considered in Sec. 7.1. – grids 327789 ; ■ grids 410375 ; ○ grids 464275 154

7.4 Comparison of the predicted velocity profiles $u_\theta(r,y)$ with other two solutions obtained at different angles θ defined in Fig. 7.1 (a) $y = 0.25$ plane ; (b) $y = 0.5$ plane. 155

7.5 Comparison of the predicted axial velocity profiles $u_\theta(r,y)$ with other two solutions obtained at different angles θ defined in Fig. 7.1 156

7.6 Pressure contours plots at three different planes 156

7.7 Velocity and pressure contours within the elbow region. 157

7.8 Velocity vector plots at different planes for the 90-degree bend duct flow problem considered in Sec. 7.1 158

7.9 Schematic of the three-dimensional backward-facing step flow problem considered in Sec. 7.2 159

7.10 Residual reduction plots for the three-dimensional backward-facing step flow problem considered in Sec. 7.2. (a) $Re = 100$; (b) $Re = 389$ 160

7.11 Comparisons of the streamwise velocity profiles for the three-dimensional backward-facing step flow problem considered in Sec. 7.2. (a) $Re = 100$; (b) $Re = 389$; (c) $Re = 1,000$ 161

7.12 Comparison of the reattachment lengths x_1/h with the results of Armaly et al. [119] for the three-dimensional backward-facing step flow considered in Sec. 7.2 162

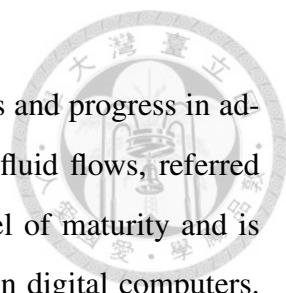


Chapter 1

Introduction

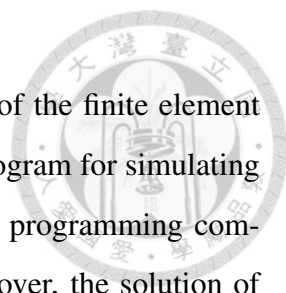
Some physical fluid flow phenomena such as pipe flow, blood flow in vessels, flow around airfoils, can be usually modeled by the three physical conservation laws : (1) mass conservation ; (2) momentum conservation and (3) energy conservation. These physical laws can be formulated in terms of mathematical equations, which in most general form are partial differential equations [1–3]. To investigate the fluid flow system, one can use the theoretical or experimental method. In the theoretical method, the closed-form analytical solution of the fluid flow equations is sought under some assumptions and simplifications in order to make the problem solvable. A closed-form solution is highly desired because the variation of flow phenomena with change in the controlling parameter is explicitly displayed. Unfortunately, the general closed-form analytical solution for most flow problem is still not available, even for a relatively idealized physical fluid dynamics system.

Use of experimental methods involve equivalent setup, often scaled down from the real physical problem to investigate the flow phenomena. The primary advantage of experimental methods is that it represents a true reality. It, therefore, is expensive to conduct measurement. The wind tunnel, for example, as a piece of experimental equipment, provides an effective way to investigate the real flow phenomena in aerodynamics. Traditionally, this has provided a cost effective alternative to full-scaled measurement. However, the design and construction of measurement equivalent that depends critically on the flow behavior, e.g. the design of airplane, full- scaled measurement as part of the design process is economically impractical [4].



With the advent of ever improving digital computers since 1950s and progress in advanced numerical analysis, numerical methods for the analysis of fluid flows, referred to as Computational Fluid Dynamics (CFD) [5] has reached a level of maturity and is now accepted as an useful tool to simulate fluid flow phenomena on digital computers. It replaces the governing equations of fluid flow in a discrete, digital form, and advances these discrete representations in space and/or time to obtain a final numerical description of the complete flow field of interest. This branch of fluid mechanics complements the experimental and theoretical method. It provides an alternative cost-effective means of simulating the real flow behavior. It has been used for the basic study of fluid dynamics, for engineering designs in complex flow configuration, for understanding and predicting fluid-structure interaction behavior, for interpreting and analyzing experimental data, and for extrapolation into parameter regimes that are relatively inaccessible or very costly to study experimentally. Each of these three methods has its own inherent strength and weakness, none of them can be dispensable for getting a complete understanding of the real physical flow system. The theoretical method is still practical and the experimental method will continue to be conducted.

The fundamental idea of CFD is to partition the spatial domain into a finite number of non-overlapping sub-domains which are called control volumes or elements. Time is also divided into small discrete lengths called time steps. The governing fluid equations are then replaced with the algebraic equations which in turn are solved to obtain the time-dependent or steady-state solutions on these subdomains. Underlying this idea, various numerical techniques such as finite difference method (FDM), finite element method (FEM), finite volume method (FVM) and Lattice Boltzmann method (LBM) have been developed by engineers and mathematicians in the past three decades. The finite element method, which can be tracked back to the 1940s [6], is recognized as an important numerical method for solving incompressible viscous flow and heat transfer problems because of its appealing advantages in treating complicated geometry and implementation of natural boundary condition. It is also mathematically rich in providing the analysis of conver-

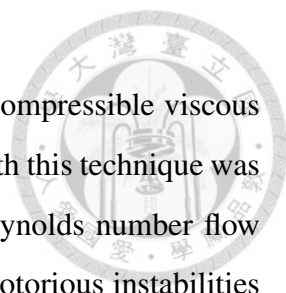


gence proof [7, 8]. Furthermore, the clear structure and versatility of the finite element method make it possible to construct a general purpose computer program for simulating various flow problems. The disadvantages arise from its increased programming complexity and more memory demands in large-sized problems. Moreover, the solution of associated algebraic system is the most prohibitively computationally intensive and parallel computation [9] is therefore required to accelerate the computation. The main goal of this dissertation is to address the calculation of the finite element method for the incompressible Navier-Stokes equations. This chapter includes reviews of finite element method, the implementation of parallel computation, the objectives, and the outline of this dissertation.

1.1 Finite element method

The finite element method is a numerical method for seeking the approximate solutions of partial differential equations in science and engineering. This method was originally introduced by engineers in the 1940s and now it is widely used in different areas in science and engineering, including the mechanical engineering, structural design, biomechanics, electromagnetic and fluid dynamics and other areas. The underlying ideas of the finite element method are variational principle and piecewise interpolation approximations [10]. The discretization process of the finite element method starts from the reformulation of the given differential equation as an equivalent variational problem. Then, the problem domain is divided into the several non-overlapping elements. These elements can be chosen as triangle/quadrilateral or tetrahedral/hexahedral, depending on the problem dimensions. The solution of differential equations is approximated on each element by using the locally defined interpolation functions. Thus, the finite element method differs from the traditional weighted-residuals methods (e.g. Rayleigh-Ritz, least-square, collocation) in the manner in which the interpolation functions are chosen globally on entire problem domain. The reader can refer to some introductory textbooks [11, 12].

In the beginning, the finite element method was applied to solve the solid mechanics



and heat transfer problems. The first finite element modeling of incompressible viscous flow can be tracked back to the 1970s. However, the early success with this technique was only limited to the low-Reynolds number problem. In the high Reynolds number flow problem in which the convection effect becomes dominated, some notorious instabilities problems are encountered. To overcome this difficulty, a series of the so-called stabilized finite element models [13, 14] have been developed in the past three decades.

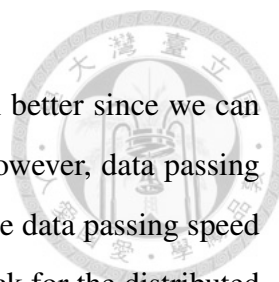
1.2 Parallel computation

The traditional way of executing computer program to solve differential equations is known as serial computing. Several tasks are performed sequentially, thus one task can start at the end of the previous one. An alternative to this mode is to perform several tasks simultaneously by using several processor units, which is known as parallel computation [9]. Large-sized problem can be therefore divided into many smaller ones, which are then executed in parallel. Currently, parallel computation is a common strategy for compute-intensive simulations

The two major computer configurations which execute the parallel computation are the shared memory and the distributed memory configurations. In computer science, the shared memory configuration refers to multiprocessing computer system and the distributed memory configuration refers to multiple-processors computer system.

For the shared memory configuration, each processor can save/load the same memory space, and the data passing can be done just in memory. Shared memory can be accessed simultaneously by one or multiple programs. The advantage of this configuration is its relative ease to program since all processors shared the same data and the communications between processors can be done within a very short time. However, the bottleneck is its scalability because the memory and processor can not be easily extended.

For the distributed memory configuration, each processor has its own private memory space. Computational tasks can be only executed on local data and the data passing is performed through high-speed network. Comparing with the shared memory config-



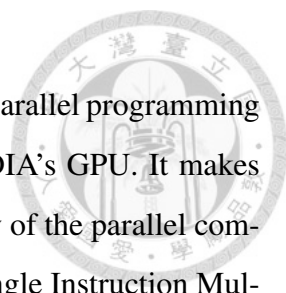
uration, the scalability in distributed memory configuration is much better since we can easily add a new processor to distributed memory configuration. However, data passing speed is much slower than the shared memory configuration since the data passing speed of network is always slower than the memory passing. The bottleneck for the distributed memory configuration is the data passing speed of network.

Over the last decade, a new alternative for parallel computation has emerged, known as Graphic Processing Unit (GPU). The availability of both the GPU hardware and programmable software has attracted the attention of researcher. Executing computer program on GPU is current practice nowadays, but depending on the data parallelism.

Parallel computations can be mainly implemented in three different ways : (i) MPI for shared and distributed memory configurations (ii) OpenMP for shared memory configuration and (iii) CUDA for NVIDIA's graphic processing units (GPUs).

MPI (Message Passing Interface) is a library which consists of a set of message passing routines. It is based on the consensus of the MPI Forum, which has over 40 participating organizations, including vendors, researchers, software library developers, and users. The goals of the MPI is to establish a portable, effective and flexible standard for programming the message passing program in Fortran, C/C++ programming languages. The MPI program is executed by the high level API (Application Passing Interface) that allows programmers to transparently make use of multiple processors on both shared or distributed memory configurations. The programmer does not need to know with the details of the communication protocol between processors.

OpenMP (Open Multi-Processor) was first released in 1997 and has been a standard API that supports shared memory parallel programming in Fortran, C and C++. It consists of a set of compiler directives and library routines that affect run-time behavior. The significant advantages of OpenMP are its portable, scalable and easy implementation on the existing computer codes. It also has the advantages of being widely used and ideally suited to multi-core architectures. The reader can refer to MPI and OpenMP programm guide for more details [15]



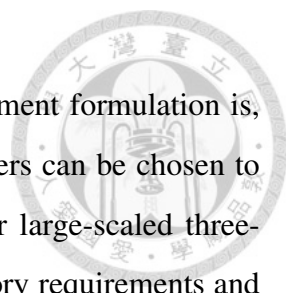
CUDA (Compute Unified Device Architecture) [16] stands for a parallel programming model released by NVIDIA in 2007 and is executed only on NVIDIA's GPU. It makes programmer directly access to the virtual instruction set and memory of the parallel computational cores in GPU. CUDA programming model is based on Single Instruction Multiple Data (SIMD) framework and provides a set of parallel instructions such that each core works on different data but executes the same instruction. The significant feature of the CUDA is that it always involves two different computation platforms : the serial code on CPU and the parallel code on GPU. GPU usually takes over the most expensive computational tasks. In this dissertation, the CUDA parallel computation approach will be chosen to accelerate finite element computations.

1.3 Objectives

The convection term in the absence of strong physical diffusion effect is the most difficult problem to simulate and this is pacing limitation in CFD techniques. The attempts to circumvent this primary instability difficulty have spawned the extensive development of CFD.

In many physical problems, ability of predicting the transport field variable is a very important and sometimes is even a crucial issue in many processing industries. Thus, the study of the transport equation is the subject of fundamental importance because this equation has been viewed as the linearized form of Navier-Stokes equations in the development of numerical scheme to solve the fluid, mass and heat transfer problem. This equation is also of considerable academic interest because of the available analytical solution, which provides a convenient verification for verifying the developed numerical scheme. For the all investigated transport equation and incompressible Navier-Stokes equations, the first-order derivative term is appeared in these equations. The minimum of wavenumber error idea can be adopted to preserve the dispersive nature for all the calculations.

When solving the primitive variable form of incompressible Navier-Stokes equations,



the resulting matrix equations discretized from the mixed finite element formulation is, in general, unsymmetric and indefinite. The direct or iterative solvers can be chosen to solve the matrix equations. However, the cost of direct solver for large-scaled three-dimensional problem is prohibitive due to its large amount of memory requirements and the iterative solver is thus adopted [17]. However, the poor convergence behavior or even divergence is often an unavoidable outcome when using the iterative solver. This problem is particularly severe at high Reynolds number. To circumvent this difficulty, one can consider solving the equivalent symmetric and positive definite normal matrix equations by using the robust iterative solver. This strategy, however, results in the increase of condition number and the pre-conditioning can be used to reduce the condition number.

The objective of this dissertation is to effectively solve the 2D/3D high Reynolds incompressible Navier-Stokes equations. To this end, we will develop a new streamline upwind finite element model. This model minimizes the wavenumber error of the convection term and eliminates the oscillatory velocity solutions in the convection-dominated problem. In order to get the unconditionally convergent solution from the finite element matrix equations, we will solve the equivalent symmetric and positive normal matrix equations. The pre-conditioners are also presented to reduce the condition number.

It is known that the finite element calculation is a time-consuming computational task. This is particularly significant for three-dimensional problem. To accelerate the calculation, we will execute the finite element calculation on a hybrid CPU/GPU platform and evaluate the advantages of its implementation on a GPU over that of its CPU sibling.

1.4 Outlines of this study

The present work is focused on developing the finite element model and executing the finite element calculations on a hybrid CPU/GPU platform. In Chapter 2, some different theoretical formulations of the governing incompressible Navier-Stokes equations, solution algorithm and computation challenges are introduced. The use of the developed finite element model for solving the scalar transport equation and incompressible Navier-Stokes

equations will be presented in Chapters 3-4. Chapter 5 assesses the performance of iterative solver for three-dimensional incompressible Navier-Stokes equations. To accelerate the finite element calculations, the CUDA parallel computation and some implementation details are presented in Chapter 6. In Chapter 7, the developed finite element CPU/GPU fluid solver was applied to investigated some practical flow problems. Finally, some remarks are concluded in Chapter 8.



Chapter 2

Incompressible Navier-Stokes equations

The Navier-Stokes equations developed by Navier [18] and Stokes [19] have been investigated rather extensively over the last few decades because they can be applicable to model fluid flow in numerous scientific and engineering applications. These equations arise from applying Newton's second law to fluid motion, together with the assumption that the stress in the fluid is the sum of a diffusive viscous term and a pressure. According to the chosen working variables, the incompressible Navier-Stokes equations can be cast into the primitive and non-primitive variable formulations. Each of these formulations has its own advantages and disadvantages. In this chapter, the introduction of three different mathematical formulations, including the primitive velocity-pressure, non-primitive vorticity-streamfunction and velocity-vorticity formulations, are shortly given. In addition, some common solution algorithms are also introduced.

2.1 Velocity-pressure variables formulation

Let Ω be an open and a bounded domain in \mathbb{R}^n (spatial dimension $n = 2, 3$) and $\partial\Omega = \Gamma$ denotes its boundary. The incompressible Navier-Stokes equations cast in primitive velocity vector $\underline{\mathbf{u}}$ and scalar pressure field p consist of the momentum conservation equations which are the mathematical expressions of the Newton's second law in motion

$$\rho \frac{\partial \underline{\mathbf{u}}}{\partial t} + \rho \underline{\mathbf{u}} \cdot \nabla \underline{\mathbf{u}} = -\nabla p + \mu \nabla^2 \underline{\mathbf{u}} + \underline{\mathbf{f}} \quad (2.1)$$

and the continuity equation which represents the conservation of mass

$$\nabla \cdot \underline{\mathbf{u}} = 0 \quad (2.2)$$

In the above, ρ is the density and μ is the dynamic viscosity. The details of the derivation of these equations can be found in [1–3]. Note that the continuity equation or called the incompressible constrain condition becomes a constraint on the velocity field $\underline{\mathbf{u}}$, and the pressure p is treated implicitly as the gradient form.

In general, it is convenient to rewrite the Eqs. (2.1)-(2.2) as a dimensionless form by defining the following dimensionless variables

$$\mathbf{x}^* = \frac{\mathbf{x}}{L}, \mathbf{y}^* = \frac{\mathbf{y}}{L}, \underline{\mathbf{u}}^* = \frac{\underline{\mathbf{u}}}{\underline{\mathbf{u}}_\infty}, \mathbf{p}^* = \frac{p}{\rho \underline{\mathbf{u}}_\infty^2}, \underline{\mathbf{f}}^* = \frac{\underline{\mathbf{f}}L}{\underline{\mathbf{u}}_\infty^2}, t^* = \frac{\underline{\mathbf{u}}_\infty t}{L} \quad (2.3)$$

where L and $\underline{\mathbf{u}}_\infty$ are the user-specific characteristic length and velocity. Substituting Eq. (2.3) into Eqs. (2.1)-(2.2) and omitting the asterisk symbol, the dimensionless form of incompressible Navier-Stokes equations is obtained

$$\frac{\partial \underline{\mathbf{u}}}{\partial t} + \underline{\mathbf{u}} \cdot (\nabla \underline{\mathbf{u}}) + \nabla p - \frac{1}{Re} \nabla^2 \underline{\mathbf{u}} = \underline{\mathbf{f}} \quad (2.4)$$

$$\nabla \cdot \underline{\mathbf{u}} = 0 \quad (2.5)$$

In Eq. (2.4), the Reynolds number ($Re = \rho \underline{\mathbf{u}}_\infty L / \mu$) represents a ratio between the inertia and viscous forces. The set of variables ($\underline{\mathbf{u}}, p$) is called the primitive variables since they can be measured directly in experiments. Moreover, other variables in a fluid flow such as streamfunction or vorticity can be calculated from these primitive variables.

In order to make the partial differential equations Eqs. (2.1)-(2.2) be well posed, it is necessary to prescribe the boundary condition on the boundary of physical domain. For this reason, either the Dirichlet-type boundary condition on Γ_D

$$\underline{\mathbf{u}} = \underline{\mathbf{g}}$$

where

$$\int_{\Gamma_D} \underline{\mathbf{n}} \cdot \underline{\mathbf{g}} d\Gamma_D = 0$$

or the Neumann-type boundary condition on an open boundary Γ_N is specified with an unit outer normal vector \underline{n}

$$-\underline{p}\underline{n} + \frac{1}{Re}\underline{n} \cdot \nabla \underline{u} = \underline{h}$$

The \underline{h} denotes an external traction boundary force. The boundary subdomains Γ_D and Γ_N , which are assumed to be non-overlapping, span the whole boundary in the following sense

$$\Gamma = \Gamma_D \cup \Gamma_N$$

$$\Gamma_D \cap \Gamma_N = \emptyset$$

When the fluid is in contact with the wall boundary, the velocity boundary value of fluid is equal to the velocity of wall. The conditions on the tangent components of velocity is known as no-slip condition. Note that no pressure boundary condition is required on no-slip boundary. This means that it would be incorrect to impose pressure value together with the velocity boundary condition. On the other hand, in some problem, such as on the inflow or outflow boundary, the velocity conditions different from Dirichlet type. In these situations, the pressure can be supplemented by Dirichlet or Neumann type condition

A fundamental observation associated with Eq. (2.1) is that the momentum equations should be used to approximate the velocity field while the continuity equation models the equation for the pressure. It seems rather natural since Eq. (2.1) is solved for the velocity components, and this leaves only Eq. (2.2) to obtain the pressure solution. This formulation, however, encounters a serious problem because the absence of the pressure term in Eq. (2.2) will destabilize the partial differential system from the computational standpoint.

2.2 Streamfunction-vorticity formulation

The absence of pressure term in Eq. (2.4) and the divergence-free constraint condition prompted researchers to seek another formulation which contains no pressure variable and can automatically satisfy the Eq. (2.5). The first and probably the most successful





one is the use of vorticity-streamfunction formulation.

Consider the velocity field $\underline{u} = (u, v)$ in two-dimensions and the divergence-free constraint condition

$$u_x + v_y = 0 \quad (2.6)$$

The streamfunction ψ is defined as follows

$$u = \psi_y, \text{ and } v = -\psi_x \quad (2.7)$$

It is clear that the Eq. (2.7) automatically satisfies the Eq. (2.6). Thus, if we find a governing equation for the ψ , we can obtain the divergence-free velocity field via Eq. (2.7).

Recall the definition of vorticity

$$\underline{\omega} = \nabla \times \underline{u}, \quad (2.8)$$

Note that Eq. (2.8) only has a single non-zero component in two-dimensions given below

$$\omega = v_x - u_y \quad (2.9)$$

Substituting Eq. (2.7) into Eq. (2.9), one can obtain the Poisson equation for streamfunction ψ

$$\nabla^2 \psi = -\omega$$

To find the governing equation for the ω , one can take the curl of Eq. (2.4) to get

$$\nabla \times \frac{\partial \underline{u}}{\partial t} + \nabla \times \left[\frac{\partial \underline{u}}{\partial t} + (\underline{u} \cdot \nabla) \underline{u} - \frac{1}{Re} \nabla^2 \underline{u} + \nabla p \right] = \nabla \times \underline{f} \quad (2.10)$$

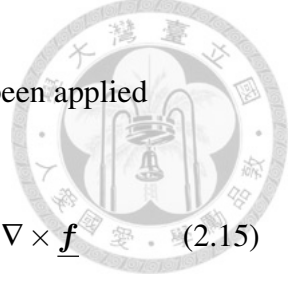
Now, the following identities are used

$$(\underline{u} \cdot \nabla) \underline{u} = \frac{1}{2} \nabla |\underline{u}|^2 - \underline{u} \times (\nabla \times \underline{u}) \quad (2.11)$$

$$\nabla \times \nabla |\underline{u}|^2 = 0 \quad (2.12)$$

$$\nabla \times \nabla p = 0 \quad (2.13)$$

$$\nabla \times (\nabla \times \underline{u}) = \nabla(\nabla \cdot \underline{u}) - (\nabla \cdot \nabla) \underline{u} = \nabla(\nabla \cdot \underline{u}) - \nabla^2 \underline{u} = -\nabla^2 \underline{u} \quad (2.14)$$



In Eq. (2.14), the divergence-free constrain condition $\nabla \cdot \underline{u} = 0$ has been applied

Substituting Eqs. (2.11)-(2.14) into Eq. (2.10) yields

$$\nabla \times \underline{u}_t - \nabla \times [\underline{u} \times (\nabla \times \underline{u})] - \frac{1}{Re} \nabla \times \nabla \times (\nabla \times \underline{u}) = \nabla \times \underline{f} \quad (2.15)$$

Using Eq. (2.8), one can get

$$\nabla \times \frac{\partial \underline{u}}{\partial t} - \nabla \times (\underline{u} \times \omega) - \frac{1}{Re} \nabla \times \nabla \times \omega = \nabla \times \underline{f} \quad (2.16)$$

The vorticity transport equation (2.16) can be simplified via Eq. (2.14)

$$\frac{\partial \omega}{\partial t} + (\underline{u} \cdot \nabla) \omega - (\omega \cdot \nabla) \underline{u} - \frac{1}{Re} \nabla^2 \omega = \nabla \times \underline{f} \quad (2.17)$$

In the above, the third vorticity stretching term represents the generation or destruction of vorticity due to the stretching or compression of vortex lines. In two-dimensional problem, the vortex stretching term vanishes and the resulting vorticity transport equation is reduced to a scalar convection-diffusion equation for the vorticity component

$$\frac{\partial \omega}{\partial t} + (\underline{u} \cdot \nabla) \omega - \frac{1}{Re} \nabla^2 \omega = \nabla \times \underline{f} \quad (2.18)$$

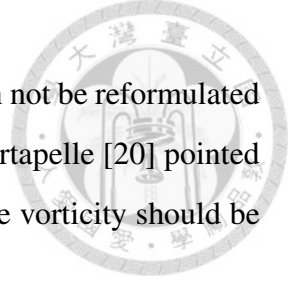
The Eq. (2.18) is the desired governing equation for the ω . The two-dimensional incompressible Navier-Stokes equations can be then rewritten into the following set of coupled scalar transport equations

$$\frac{\partial \omega}{\partial t} + (\underline{u} \cdot \nabla) \omega - \frac{1}{Re} \nabla^2 \omega = \nabla \times \underline{f} \quad (2.19)$$

$$\nabla^2 \psi = -\omega \quad (2.20)$$

The above (ω, ψ) formulation offers the advantages of reducing the number of unknowns and eliminating the divergence-free constraint condition which is difficult to be preserved numerically. Moreover, the pressure term is eliminated in the Eqs. (2.19)-(2.20) and we do not need to prescribe the pressure boundary.

The $(\underline{\omega}, \psi)$ formulation, however, still encounters some difficulties. Firstly, this formulation can not be applied to three-dimensional problem since the streamfunction ψ is only available in two-dimensions. Secondly, there is no available explicit boundary condi-



tion for the vorticity ω since the no-slip condition for the velocity can not be reformulated in equivalent condition of boundary value type for the vorticity. Quartapelle [20] pointed out that to satisfy the no-slip boundary condition for the velocity, the vorticity should be subject to an integral constraint equation.

2.3 Velocity-vorticity formulation

The third representation of the incompressible Navier-Stokes equations involves the primitive velocity variable \underline{u} and the non-primitive vorticity variable $\underline{\omega}$. By taking the curl of both sides of Eq. (2.4) and using Eqs. (2.11)-(2.14), we can obtain vorticity transport equation as follows

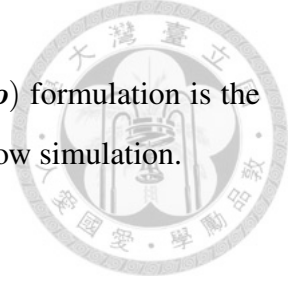
$$\frac{\partial \underline{\omega}}{\partial t} + (\underline{u} \cdot \nabla) \underline{\omega} - (\underline{\omega} \cdot \nabla) \underline{u} - \frac{1}{Re} \nabla^2 \underline{\omega} = \nabla \times \underline{f} \quad (2.21)$$

By taking the curl of Eq. (2.8) and using Eq. (2.2), we get

$$\nabla^2 \underline{u} = -\nabla \times \underline{\omega} \quad (2.22)$$

which is the vector form of Poisson equation for the velocity \underline{u} . Eqs (2.21)-(2.22), with \underline{u} and $\underline{\omega}$ as velocity and vorticity vectors, are known as the velocity-vorticity formulation of the incompressible Navier-Stokes equations. The proper boundary condition for vorticity is also required to be prescribed to get the unique solution of coupled system. Note that the boundary and integral conditions enable the velocity-vorticity formulation to be adopted in two and three dimensions.

The two-dimensional incompressible Navier-Stokes equations can be expressed in terms of (1) primitive variable (\underline{u}, p) formulation ; (2) Non-primitive variables (ψ, ω) formulation and (3) hybrid variables (\underline{u}, ω) formulation. In this dissertation, we advocate adopting the primitive (\underline{u}, p) formulation hereafter in the incompressible flow calculations. The great advantage of this formulation over other formulations is that the variable setting resolves an ambiguity in specifying legitimate boundary condition. In addition, no artificial boundary condition needs to be prescribed. Eqs. (2.4) -(2.5) are subject only to boundary velocities. Specification of pressure conditions at the physical boundary will



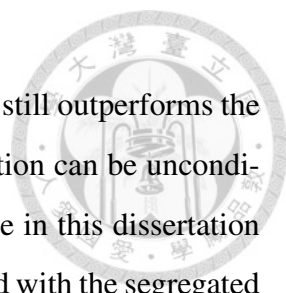
overdetermine the differential system. Moreover, the primitive (\underline{u}, p) formulation is the simplest and has the potential of extending it to three-dimensional flow simulation.

2.4 Solution algorithms

The methods for solving the primitive variable form of the incompressible Navier-Stokes equations can be mainly categorized into two groups, depending on how the primitive variables (\underline{u}, p) are treated. They are the mixed coupling and the segregated uncoupling approaches. In the mixed coupling approach, the discretized Eqs. (2.4)-(2.5) are solved simultaneously, treating all the dependent variables (\underline{u}, p) as the unknowns. This is correct and easy to implement. Moreover, the coupling between the velocity and pressure can be retained, a small number of nonlinear iterations is required to obtain the convergent solution. However, this approach is rather time consuming and demands a large amount of memory requirement.

To improve the computational efficiency, the segregated uncoupling approach, such as the SIMPLE-family methods [21–24], projection-family methods [25, 26], characteristic based splitting (CBS) methods [27, 28], were presented in the past three decades. The distinguish feature of this approach is the use of a derived equation to determine the pressure field. Typically, the momentum equations for the velocity components are solved separately, using the available values of other variables. The resulting velocity solution does not, in general, satisfy the divergence-free constrain condition Eq. (2.5) and it is necessary to be corrected. Then the governing equation for pressure field derived from Eq. (2.4) is solved to get the pressure solution. Finally, the velocity field is corrected using the pressure solution. This procedure is repeated until the user specific convergence criterion is satisfied.

The mixed and segregated approaches have emerged as being among the most popular classes of methods to solve Eqs. (2.4)-(2.5). The mixed approach has found some applications in two-dimensional problems. An extension of this approach to three-dimensional problem suffers from the calculation of large-scaled matrix equations. This presents ob-



stacles to extend the application scope. Even so, the mixed approach still outperforms the segregated counterpart because the divergence-free constraint condition can be unconditionally satisfied. This is an appealing advantage and we concentrate in this dissertation on the mixed approach, even through it is memory intensive compared with the segregated approach.

2.5 Computation challenges

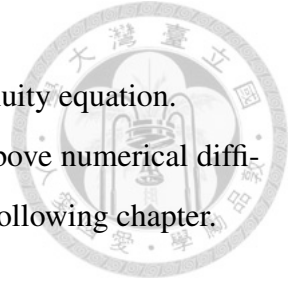
As for the incompressible Navier-Stokes equations, we encounter the dilemma of having to reconcile solution accuracy and numerical stability. It is often the case that a gain in solution accuracy usually accompanies with a lose in stability. In the high Reynolds number problem, the upwinding scheme serves as a common cure for the enhancement of instability arising from the convection term, they are accompanied with a deteriorated accuracy.

Aside from the suppression of the oscillatory velocity solutions, three other notorious difficulties commonly encountered are : (i) The absence of pressure term in the momentum equations ; (ii) The elimination of the node-to-node decoupling of the pressure solution and (iii) The satisfaction of divergence-free constrain condition to preserve the mass conservation.

As mentioned in previous section, we advocate adopting the mixed coupling approach to solve Eqs. (2.4)-(2.5) due to its unconditional satisfaction of continuity equation. The approach, in general, results in an unsymmetric and indefinite matrix equations. These properties pose an another challenge. The Gaussian elimination-based direct solver has long been considered to be the only means of resolving these difficulties. However, the demands for a continued fill-in the course of finding a three-dimensional solution using direct solver is beyond the scope of computer memory. The cost of direct solver is too expensive to be applied and one has to use the iterative solver for large-sized three-dimensional problem. Poor convergence or divergence is, therefore, an unavoidable outcome when using the iterative solver. This problem is associated particularly with the

mixed finite element formulation, owing its direct handling of continuity equation.

In this dissertation, we will present the means of resolving the above numerical difficulties. Furthermore, many of these results will be presented in the following chapter.





Chapter 3

Finite element model for the transport equation

3.1 Introduction

In many physical problems, such as the pollutant dispersal in a river estuary or the concentration of electrons in modeling semiconductor devices, involve a combination of convective and diffusive processes [29]. Such a transport is a very important and sometimes is even a crucial process for designers. Thus, the partial differential equation which governs the convective and diffusive transport processes is of great importance in the fields of fluid mechanics and heat/mass transfer. This equation is even regarded as important itself from the numerical scheme point of view. The real importance lies in its resemblance to the linearized form of Navier-Stokes equations. Simply stated, this equation can be viewed as the simplified Navier-Stokes equations and is thus the subject of interest, academically as well as practically. Moreover, it is amenable to analytical solution and provides a convenient validation feat bed for benchmarking the developed numerical discretization scheme.

3.2 Governing equation

Let Ω be an open and bounded domain, the following transport equation is investigated

$$\frac{\partial \phi}{\partial t} - \varepsilon \Delta \phi + \underline{u} \cdot \nabla \phi = f \text{ in } \Omega \quad (3.1)$$

The Eq. (3.1) is called the convection-diffusion or scalar transport equation, in which ϕ represents some physical quantities be transported (e.g., chemical concentration, temperature or velocity). Here $\varepsilon > 0$ is a constant diffusion coefficient, \underline{u} is a given velocity-valued convection field with $\nabla \cdot \underline{u} = 0$, and f is a given source term. The essential and natural boundary conditions are given by

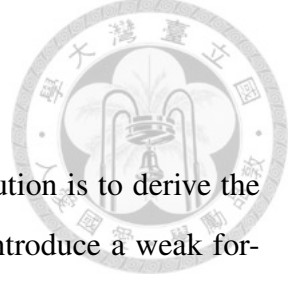
$$\phi = g_1 \text{ on } \Gamma_D \quad (3.2)$$

$$\varepsilon \nabla \phi \cdot \underline{n} = g_2 \text{ on } \Gamma_N \quad (3.3)$$

where Γ_D and Γ_N represent the Dirichlet and Neumann parts of the boundary $\partial\Omega = \Gamma (= \Gamma_D \cup \Gamma_N)$, g_1 and g_2 are the prescribed values and \underline{n} is the unit outward normal vector.

In general, the diffusion coefficient is relatively small compared with the magnitude of convection field \underline{u} . For example, for a smoke ejecting from a chimney moves following the direction of wind and spreading due to molecular diffusion which is relative small. This implies that for most practical situations, $\varepsilon \ll |\underline{u}|$.

To characterize the relative importance of the convective and diffusive effects, the dimensionless mesh Peclet number defined as $Pe = \frac{|\underline{u}|H}{2\varepsilon}$, where H represents the element length, is adopted to express the ratio of convective to diffusive effect. Provided that Eq. (3.1) is convection-dominated, the mesh Peclet number is larger than 2. Under this situation, the classical Galerkin finite element model, in which the trial and weighting function are chosen from the same functional space, performed very poorly since large node-to-node oscillations exhibit not only near the boundary or in interior layers but also in other regions. These oscillations seriously reduce the solution accuracy and can even make the computed results to be meaningless. To overcome this difficulty, a large class of the so-called stabilized finite element models has been developed and intensively studied in the past three decades [13].



3.3 Weak formulation

As mentioned before, the first step of finding the finite element solution is to derive the weak formulation of the targeted partial differential equation. To introduce a weak formulation, the standard notations for the $\mathcal{L}^2(\Omega)$ space and its associated innerproduct (\cdot, \cdot) and norm $\|\cdot\|$ are given below.

$$\begin{aligned}\mathcal{L}^2(\Omega) &= \{\underline{\mathbf{u}} \mid \int_{\Omega} \underline{\mathbf{u}}^2 d\Omega < \infty\} \\ (\mathbf{p}, \mathbf{q}) &= \int_{\Omega} \mathbf{p}\mathbf{q} d\Omega, \quad \|\mathbf{p}\| = (\mathbf{p}, \mathbf{p})^{1/2}\end{aligned}$$

The $\mathcal{L}^2(\Omega)$ denotes the space of scalar valued functions that are square-integrable on Ω . In addition, the $\mathcal{H}^1(\Omega)$, which denotes the space of vector-valued functions with square-integrable derivative on Ω , is also given

$$\mathcal{H}^k(\Omega) = \{q \in \mathcal{L}^2(\Omega) ; D^s q \in \mathcal{L}^2(\Omega), s = 1, \dots, k\} \quad (3.4)$$

where D^s denotes the derivative of order s . As usual, $\mathcal{L}(\Omega) = \mathcal{H}^0(\Omega)$ and $\mathcal{H}^1(\Omega)$ is called the Sobolev space. For the boundary valued problem, the constrained functional spaces are defined below

$$\begin{aligned}\mathcal{H}_0^1(\Omega) &= \{\mathbf{u} \mid \mathbf{u} \in \mathcal{H}^1(\Omega), \mathbf{u} = 0 \text{ on } \Gamma\} \\ \mathcal{L}_0^2(\Omega) &= \{\mathbf{u} \mid \mathbf{u} \in \mathcal{L}^2(\Omega), \int_{\Omega} \mathbf{u} d\Omega = 0\}\end{aligned}$$

Given the above functional spaces, the weak formulation of Eq. (3.1) is derived by multiplying the Eq. (3.1) by the weighting function $w \in \mathcal{H}_0^1(\Omega)$ and then integrating the equation over the domain Ω to obtain

$$\int_{\Omega} \frac{\partial \phi}{\partial t} w d\Omega - \int_{\Omega} \varepsilon \Delta \phi w d\Omega + \int_{\Omega} (\underline{\mathbf{u}} \cdot \nabla \phi) w d\Omega = \int_{\Omega} \mathbf{f} w d\Omega \quad (3.5)$$

Applying the Green's theorem [10], one can get

$$\int_{\Omega} \frac{\partial \phi}{\partial t} w d\Omega + \varepsilon \int_{\Omega} \nabla \phi \cdot \nabla w d\Omega - \varepsilon \int_{\Gamma} w \frac{\partial \phi}{\partial \underline{\mathbf{n}}} ds + \int_{\Omega} (\underline{\mathbf{u}} \cdot \nabla \phi) w d\Omega = \int_{\Omega} \mathbf{f} w d\Omega \quad (3.6)$$



Note that the third integral term over the boundary in Eq. (3.6) will vanish and the final weak formulation of Eq. (3.1) reads as : Find $\phi \in \mathcal{H}_0^1(\Omega)$ such that

$$a(\phi, \mathbf{w}) = (\mathbf{f}, \mathbf{w}) \quad \forall \mathbf{w} \in \mathcal{H}_0^1(\Omega) \quad (3.7)$$

where the bilinear form $a(\cdot, \cdot)$ is defined as

$$a(\phi, \mathbf{w}) = \left(\frac{\partial \phi}{\partial t} + \underline{\mathbf{u}} \cdot \nabla \phi, \mathbf{w} \right) + (\epsilon \nabla \phi, \nabla \mathbf{w}) \quad (3.8)$$

Let $\{\mathfrak{D}_h\}$ be a partition of domain Ω . This partition $\{\mathfrak{D}_h\}$ of Ω consists of quadrilateral element K is constructed in the usual way : the intersection of any two elements is a vortex, or an edge or empty, and $\Omega = \cup_{K \in \{\mathfrak{D}_h\}} K$. In particular, the mesh size h is defined as $h = \max\{h_K ; K \in \mathfrak{D}_h\}$

Let $\mathcal{S}^h \subseteq \mathcal{H}_0^1(\Omega)$, $\mathcal{V}^h \subseteq \mathcal{H}_0^1(\Omega)$ be the finite element subspace for trial and weighting function, respectively. The finite element solution ϕ^h is sought from the following statement : Find $\phi^h \in \mathcal{S}^h$ such that

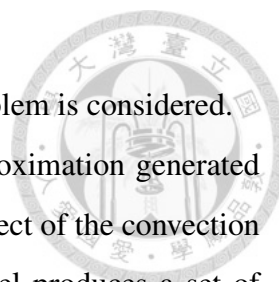
$$a(\phi^h, \mathbf{w}^h) = (\mathbf{f}^h, \mathbf{w}^h) \quad \forall \mathbf{w}^h \in \mathcal{V}^h \quad (3.9)$$

Substituting the finite element approximation $\phi^h = \sum_j \phi_j N_j$, where N_j denote the interpolation functions, and the weighting function \mathbf{w}^h into Eq. (3.9), the elementary finite element matrix equations can be calculated on each element K . These equations are then assembled to form a global algebraic linear system. By applying a proper boundary condition, the nodal values of ϕ^h are determined from the solution of matrix equations.

3.4 Finite element model

3.4.1 Literature review

The Galerkin finite element model has been proven eminently successful in problems of solid/structural mechanics and in other situations such as the heat conduction which is governed by diffusion-type equation. However, the computational difficulty is perceived by the Galerkin finite element model in fluid mechanics. The solutions of convection dominated problem are usually corrupted by non-physical node-to-node oscillations and



these oscillations become even worse when high Peclet number problem is considered.

The above difficulty was blamed on the central difference approximation generated by the Galerkin model and its inability to account for the upwind effect of the convection term. The discretization of the convection term by Galerkin model produces a set of equations that involve the adjacent nodes, and it results in node-to-node oscillations. That is, the best approximation property in the energy norm of the Galerkin model, which is the basis for success in symmetric operator, is lost when convection dominates the diffusion [14].

In principle, these oscillations can be suppressed by a successive refinement of the mesh to make the mesh Peclet number to be less than 2. However, the necessary degree of the mesh refinement is always economically impracticable. This computational difficulty and perceived shortcomings of the Galerkin model have motivated the development of the so-called Petrov-Galerkin models, in which the trial function and weighting function are chosen from the different functional spaces.

Inspired by the upwind operator in finite difference context [5], Christie [30] and Heinrich [31, 32] developed the early upwind finite element models for the steady-state one- and two-dimensional convection-diffusion equation, respectively. The weighting functions they proposed are equal to the weighting function of Galerkin model plus a high order polynomial in order to place more weight on the upwind side. Later, Hughes [33], in a different manner, modified the numerical quadrature rule for the convection term to achieve the upwind effect. Unfortunately, the success of these early Petrov-Galerkin models was only limited to one-dimensional problem because they produced in general excessive crosswind diffusion which reduces the solution accuracy in multi-dimensional problem. This problem is particularly significant in cases when mesh lines and flow directions are not in good alignment. In addition, these models are only first-order accurate and its extension to the unsteady cases or for the case of non-zero source term would not give a consistent formulation.



3.4.2 Streamline Upwind Petrov-Galerkin model

The drawbacks of the early Petrov-Galerkin models [30–33] revealed that the flow-oriented convection discretization scheme should be a better strategy to enhance convective stability without sacrificing solution accuracy. In the beginning of the 1980s, Hughes and Brook pointed out that to enhance convective stability and reduce the crosswind diffusion, it is necessary to add a proper amount of artificial damping along the direction of primary flow. The introduced stabilization term was added to the Galerkin weighting function and it is called the streamline upwind Petrov-Galerkin (SUPG) model [34]. It is also known as a perturbation method since the added stabilization term (or called the artificial damping) can be viewed as a perturbation to the Galerkin weighting function. That is, the weighting function of SUPG model is given below :

$$\mathbf{w}_{\text{SUPG}} = \mathbf{w}_{\text{Galerkin}} + \tau_{\text{SUPG}} \mathbf{B}_i \quad (3.10)$$

$$\mathbf{B}_i = \underline{\mathbf{u}} \cdot \nabla \phi \quad (3.11)$$

where \mathbf{B}_i denotes the biased part. Later, Mizukami and Hughes extended the application range of SUPG to flow problem containing a sharp layer by demanding that the discrete system underlying the streamline upwind scheme ensures the satisfying of the maximum principle [35]. In the presence of a boundary and an internal sharp layers, the predicted SUPG solution quality was deteriorated further. To overcome this shortcoming, Hughes and his colleagues improved the SUPG model by adding the discontinuity-capturing term to the SUPG weighting function so as to produce a smooth solution in the boundary or interior layer problem [36].

In Eq. (3.10), the stabilization parameter τ_{SUPG} has the significant effect on the suppression of the oscillations and the solution accuracy. The parameter τ_{SUPG} in [37] is defined as

$$\tau_{\text{SUPG}} = \frac{\delta_{\text{SUPG}} \underline{\mathbf{u}} H}{2|\underline{\mathbf{u}}|^2}$$

where H and δ_{SUPG} represent the element characteristic length ($H = 2h$) and the upwind

coefficient, respectively. In this study, the quadratic element is adopted in two- and three-dimensional problem. The upwind coefficient δ_{SUPG} will be, therefore, determined at the center and the corner node, respectively. One way to determine the coefficient δ_{SUPG} is to impose one-dimensional exactness of the homogeneous and steady-state convection diffusion equation given below [37].

$$\delta_{\text{SUPG}}(\gamma) = \begin{cases} \frac{2 - \cosh(\gamma) - (4/\gamma) \tanh(\gamma/2) + (1/\gamma) \sinh(\gamma)}{4 \tanh(\gamma/2) - \sinh(\gamma) - (6/\gamma) \sinh(\gamma) \tanh(\gamma/2)} & \text{at corner node} \\ \frac{1}{2} \coth\left(\frac{\gamma}{2}\right) - \frac{1}{\gamma} & \text{at center node} \end{cases} \quad (3.12)$$

where $\gamma = \frac{|\mathbf{u}|H}{2\varepsilon}$ represents the element Peclet number.

In this study, in the different manner, the upwind coefficient will be derived by minimizing the wavenumber error of the convection term in next section.

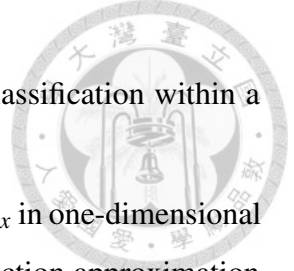
3.4.3 Modified wavenumber error optimizing model

Since the numerical instability is generated from the Galerkin finite element model treatment of the convection term, one way to enhance convective stability is to take the dispersive nature of the investigated convection (or first-order derivative) term into consideration.

A scheme for approximating the convection term can be regarded to be optimized if the error between the exact and numerical wavenumbers is minimized. This can be achieved by applying the Fourier transform $\tilde{\phi}(k) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \phi(x) \exp(-\mathbf{i}kx) dx$ and its inverse $\phi(x) = \int_{-\infty}^{\infty} \tilde{\phi}(k) \exp(\mathbf{i}kx) dk$, where $\mathbf{i} = \sqrt{-1}$, to the convection term. In this study, the stabilization parameter in modified wavenumber error (MWE) model τ_{MWE} is chosen as

$$\tau_{\text{MWE}} = \frac{\delta_{\text{MWE}}^m H}{2|\mathbf{u}|} \quad (3.13)$$

Following the work of Tam and Webb [38], derivation of the upwind coefficient δ_{MWE}^m is to minimize the wavenumber error of the convection term. As mentioned before, the nodal point in a quadratic element can be divided into the center and corner nodes, respectively.



The superscript m in Eq. (3.13) is α or β depending on the nodal classification within a quadratic element.

To minimize the wavenumber error, we take the convection term ϕ_x in one-dimensional equation as an example to determine τ_{MWE} . Substituting the trial function approximation $\phi = \sum_{j=1}^3 \phi_j N_j$ and the weighting function $w_i = N_i + \tau u \frac{\partial N_i}{\partial x}$, where N_i ($i = 1 \sim 3$) represent the quadratic interpolation functions, into the weighted residuals statement for ϕ_x , the discretized equation at the center and corner nodes can be derived as follows :

$$\phi_x|_{\text{center}} \approx \frac{1}{h}(a_1\phi_{i-1} + a_2\phi_i + a_3\phi_{i+1}) \quad (3.14)$$

$$\phi_x|_{\text{corner}} \approx \frac{1}{h}(b_1\phi_{i-2} + b_2\phi_{i-1} + b_3\phi_i + b_4\phi_{i+1} + b_5\phi_{i+2}) \quad (3.15)$$

In the above, $a_1 = -\frac{1}{2} - \delta_{\text{MWE}}^\alpha$, $a_2 = 2\delta_{\text{MWE}}^\alpha$, $a_3 = \frac{1}{2} + \delta_{\text{MWE}}^\alpha$, $b_1 = \frac{1}{4} + \frac{1}{4}\delta_{\text{MWE}}^\beta$, $b_2 = -1 - 2\delta_{\text{MWE}}^\beta$, $b_3 = \frac{7}{2}\delta_{\text{MWE}}^\beta$, $b_4 = 1 - 2\delta_{\text{MWE}}^\beta$ and $b_5 = -\frac{1}{4} + \frac{1}{4}\delta_{\text{MWE}}^\beta$. By conducting Fourier and its inverse transformation on each term shown in Eqs. (3.14), the numerical wavenumber k at the center node can be derived as $k \approx \frac{-i}{h}(a_1 \exp(-ikh) + a_2 + a_3 \exp(ikh))$. The exact wavenumber \tilde{k} is regarded as the right hand side of the above relation, thereby leading to

$$\tilde{k} = \frac{-i}{h}(a_1 \exp(-ikh) + a_2 + a_3 \exp(ikh)).$$

To demand k be close to \tilde{k} , the integral quantity $E(k)$ defined below should be a very small and positive value

$$E(k) = \int_{-\pi/2}^{\pi/2} |kh - \tilde{k}h|^2 d(kh)$$

To get the smallest value of E from the above equation, the limiting condition $\partial E / \partial a_1$ is applied to get the coefficient $\delta_{\text{MWE}}^\alpha$. The coefficient $\delta_{\text{MWE}}^\beta$ can be derived similarly. The derived upwinding coefficients are summarized below

$$\delta_{\text{MWE}} = \begin{cases} \frac{1}{2} & ; \text{ at center node} \\ \frac{8 - 3\pi}{-22 + 6\pi} & ; \text{ at corner node} \end{cases} \quad (3.16)$$

Comparing the upwind coefficients in Eq. (3.12) and Eq. (3.16), the main difference is

that the proposed stabilization parameter is a constant and the computational cost can be therefore reduced when evaluating the stabilization parameter τ_{MWE} . Another difference is that the parameter τ_{MWE} is independent of the convection term. Such a difference becomes significant when solving the nonlinear incompressible Navier-Stokes equations, where the convection velocity field will be updated in each nonlinear iteration. For the sake of clarify, the weighting functions of different finite element model are plotted in Fig. 3.1. It is clearly seen from Fig. 3.1 that the proposed MWE model gives more weight to upstream nodes than the Galerkin and SUPG model.

The extension of the above Petrov-Galerkin formulations to multi-dimensional problem is, however, obtained at the cost of accuracy. The reason is due to the use of convectional Petrov-Galerkin model which has a tendency to add a false diffusion to regions normal to the streamline as the angle between the grid line and the flow direction is large. To improve this situation, the weighting function should accommodate a streamline operator so as to provide a natural mechanism for adding the artificial damping to the weighting function along the direction of primary flow. In a quadratic element, the stabilization parameters are assigned at the center and corner nodes to stabilize the discrete system [39]

$$\tau_{\text{MWE}} = \frac{\sum_{i=1}^n \delta_{\text{MWE}}^i H^i}{2|\underline{u}|}, (n = 2 \text{ or } 3) \quad (3.17)$$

In the above, δ_{MWE} is obtained by Eq. (3.16)

3.5 Numerical studies

A finite element featured with minimized wavenumber error will be verified through the problem amenable to the analytical solution. For completeness, problems with/without interior/boundary layers are selected for use in the present study.



3.5.1 Verification study

The following steady-state transport equation for ϕ is considered first in a square domain ($0 \leq x, y \leq 1$) for the sake of code verification

$$\mathbf{u} \frac{\partial \phi}{\partial x} + v \frac{\partial \phi}{\partial y} - \frac{1}{Re} \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \right) = S. \quad (3.18)$$

In the above, Re and S denote the Reynolds number and the source term in each element, respectively. The solution ϕ is sought subject to the following analytical solutions [40].

$$u(x, y) = \frac{-2(1+y)}{(1+x)^2 + (1+y)^2} \quad (3.19)$$

$$v(x, y) = \frac{2(1+x)}{(1+x)^2 + (1+y)^2} \quad (3.20)$$

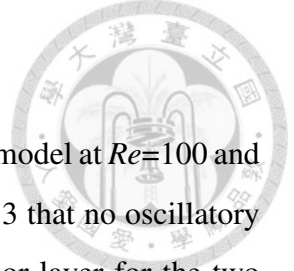
The analytical solution for ϕ has the same form as u given in Eq. (3.19) provided that $S = -\frac{\partial p}{\partial x}$, where

$$p(x, y) = -\frac{2}{(1+x)^2 + (1+y)^2} \quad (3.21)$$

All the calculations are carried out with different uniform mesh sizes to get the rate of convergence. The rate of convergence for ϕ considered at $Re = 100$ and 1000 are computed from $C = \frac{\log(Err_1/Err_2)}{\log(h_1/h_2)}$, where Err_i ($i = 1, 2$) denotes the L_2 error norm with the mesh sizes h_1 and h_2 . The predicted solutions at different mesh sizes are shown in Table 3.2. Good agreement between the solutions and the predicted rate of convergence can be seen in the result.

3.5.2 Skew problem

The benchmark skewed flow transport problem is investigated. The aim of investigating this problem is to show the effectiveness of applying the proposed finite element model to resolve high-gradient solution in the flow. The problem schematic shown in Fig. 3.2, there is a straight line, with the angle of $\theta = \tan^{-1}(v/u)$, which divides the problem domain into two subdomains. The convection field (\mathbf{u}, v) which is parallel to the dividing line for the problem with $S = 0$ in Eq. (3.18) is considered. Subject to the boundary condition setting for ϕ given in Fig. 3.2, a shear layer of high-gradient or discontinuity with the



width of $O(\sqrt{Re})$ [10] is expected when crossing the dividing line.

The finite element solutions of the proposed MWE finite element model at $Re=100$ and 1000 are obtained at $\Delta x = \Delta y = \frac{1}{60}$. It is clearly seen from Fig. 3.3 that no oscillatory solution is found to occur in regions near and apart from the interior layer for the two investigated cases.

3.5.3 Smith & Hutton problem

The second benchmark Smith & Hutton problem [41] is of some importance, for it amounts of determining whether the finite element solution is sensitive to sharply varying inlet working variable. The problem schematic shown in Fig. 3.4 will be investigated under the prescribed divergence-free convection field given by $(u, v) = (2y(1 - x^2), -2x(1 - y^2))$ and the zero source $S = 0$. The inlet condition ϕ is prescribed by $\phi(-1 \leq x \leq 0, y = 0) = 1 + \tan[10(2x + 1)]$. On the left and right boundaries, ϕ is prescribed as $1 - \tanh(10)$. Along the outlet boundary given by $(0 \leq x \leq 1, y = 0)$, a zero gradient condition for ϕ is imposed. The finite element solutions at $Re = 10^8, 10^{10}$ and 10^{12} are obtained at $\Delta x = \Delta y = \frac{1}{60}$. It is clearly seen from Fig. 3.6 that no oscillatory solution is found. The applicability of the proposed MWE finite element model for resolving the high-gradient solution is therefore confirmed.

3.5.4 Rotation shaped cone problem

The transient convection-diffusion problem schematic in Fig. 3.7(a) is then considered. For the temporal derivative term, the backward Euler scheme is employed. In the domain $-0.5 \leq x, y \leq 0.5$, the problem investigated at $(u, v) = (-4y, 4x)$ is amenable to the analytical solution given by

$$\phi(x, y, t) = \frac{2\sigma^2}{2\sigma^2 + 4\epsilon t} \exp \left[-\frac{(\bar{x} - x_c)^2 + (\bar{y} - y_c)^2}{2\sigma^2 + 4\epsilon t} \right], \quad (3.22)$$

where $(\bar{x}, \bar{y}) = (x \cos 4t + y \sin 4t, -x \sin 4t + y \cos 4t)$, $\sigma^2 = 2 \times 10^{-3}$, $\epsilon = 0$ and $(x_c, y_c) = (-0.25, 0)$. The initial condition plotted in Fig. 3.7(b) is obtained from Eq. (3.22) at $t = 0$. All the solutions computed at $\Delta t = \frac{\pi}{3200}$ and $\Delta x = \Delta y = \frac{1}{100}$ are plotted in Fig. 3.8. It is

clearly seen from the time-evolving contours at $t = \frac{\pi}{2}, \pi, \frac{3\pi}{2}$ and 2π that good agreement between the predicted and the analytical solutions is obtained. The solution symmetry can be well retained irrespective of the specified rotation convection field.



3.5.5 Mixing of warm and cold fluids problem

The transient mixing of warm and cold fluids problem schematic in Fig. 3.9(a) is investigated. In domain $-4 \leq x, y \leq 4$, this problem has the following analytical solution at the limiting case $\varepsilon = 0$ [42]

$$\phi(x, y, t) = -\tanh\left[\frac{y}{2}\cos(\omega t) - \frac{x}{2}\sin(\omega t)\right], \quad (3.23)$$

where ϕ is the temperature field and $\omega = v_t/\bar{v}_t$ denotes the rotation frequency. The $v_t = \text{sech}^2(r)\tanh(r)$ is the tangential velocity at the location that is distant from $(0,0)$ with a length r and $\bar{v}_t (= 0.385)$ is the maximum of v_t . The initial condition $\phi(x, y, t = 0)$ is plotted in Fig. 3.9(b). An initially narrow region of high gradient will be twisted by a fixed rotational velocity field $(u, v) = (-\omega y, \omega x)$. The predicted solutions are obtained at $\Delta x = \Delta y = \frac{1}{60}$ and $\Delta t = 10^{-2}$. The time-evolving contours take a spiral form and change sharply near the interface of warm and cold fluids at different times shown in Fig. 3.10. For the sake of comparison, the predicted profiles at $t = 4$ are plotted together with the analytical solution in Fig. 3.11. Good agreement of the solutions at different values of θ has been demonstrated.



grid sizes	SUPG-FEM model		MWE-FEM model	
	L_2 -error norms	R.O.C.	L_2 -error norms	R.O.C.
21×21	4.195E-6	—	7.338E-6	—
41×41	7.544E-6	2.475	1.526E-6	2.265
61×61	2.396E-7	2.828	4.626E-7	2.943
81×81	9.931E-8	3.061	2.080E-7	2.778

Table 3.1: The computed L_2 -error norms and the corresponding spatial rates of convergence (R.O.C.) for the calculations carried out at four chosen meshes using the proposed SUPG-FEM and MWE-FEM model for $Re = 100$.

grid sizes	SUPG-FEM model		MWE-FEM model	
	L_2 -error norms	R.O.C.	L_2 -error norms	R.O.C.
21×21	8.599E-6	—	3.567E-6	—
41×41	1.937E-6	2.150	4.287E-7	3.056
61×61	8.097E-7	2.151	1.433E-7	2.702
81×81	4.238E-7	2.250	7.738E-8	2.141

Table 3.2: The computed L_2 -error norms and the corresponding spatial rates of convergence (R.O.C.) for the calculations carried out at four chosen meshes using the proposed SUPG-FEM and MWE-FEM model for $Re = 1000$.

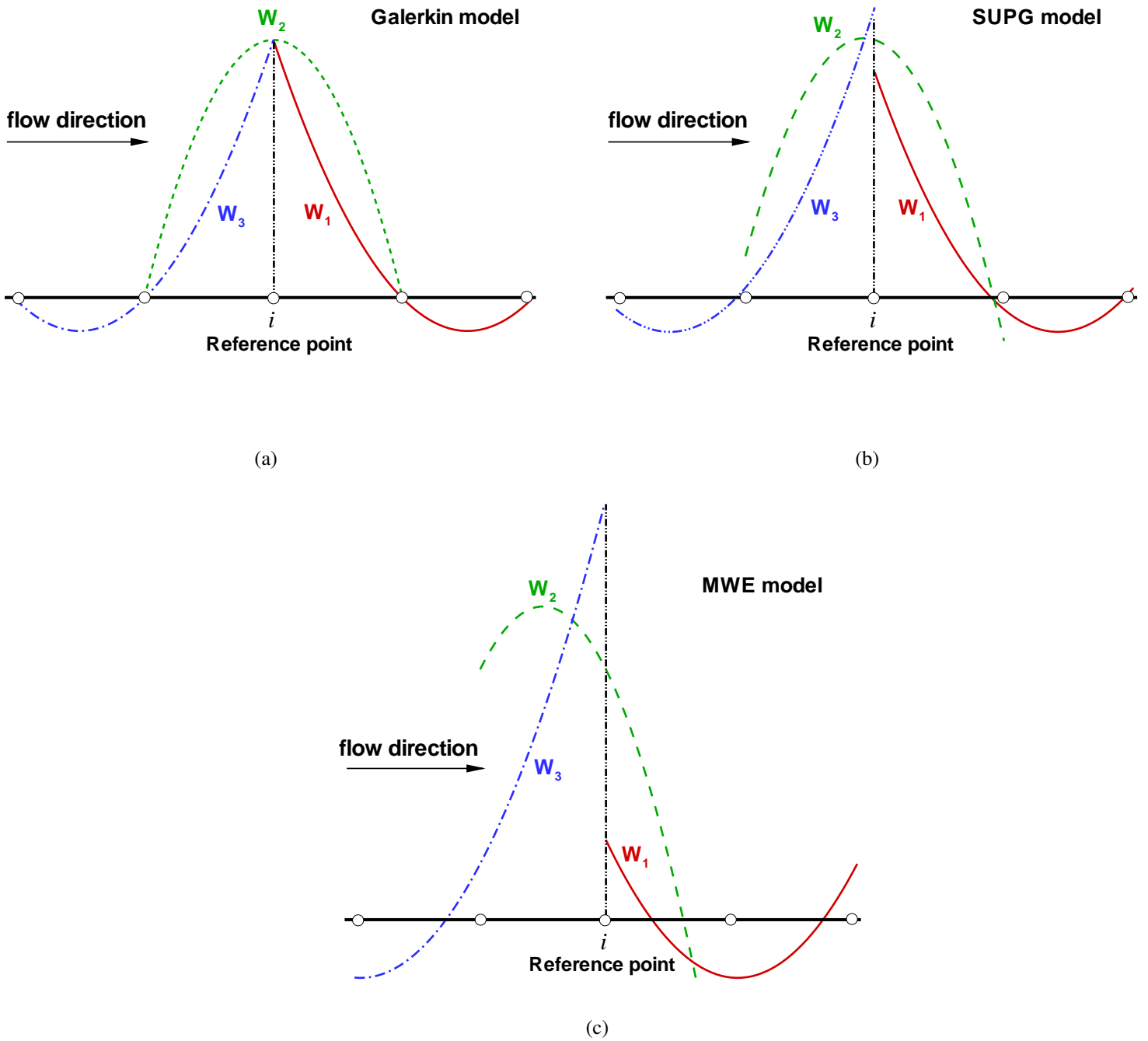


Figure 3.1: One-dimensional weighting functions plots for the different finite element model. (a) Galerkin model ; (b) SUPG model ; (c) MWE model

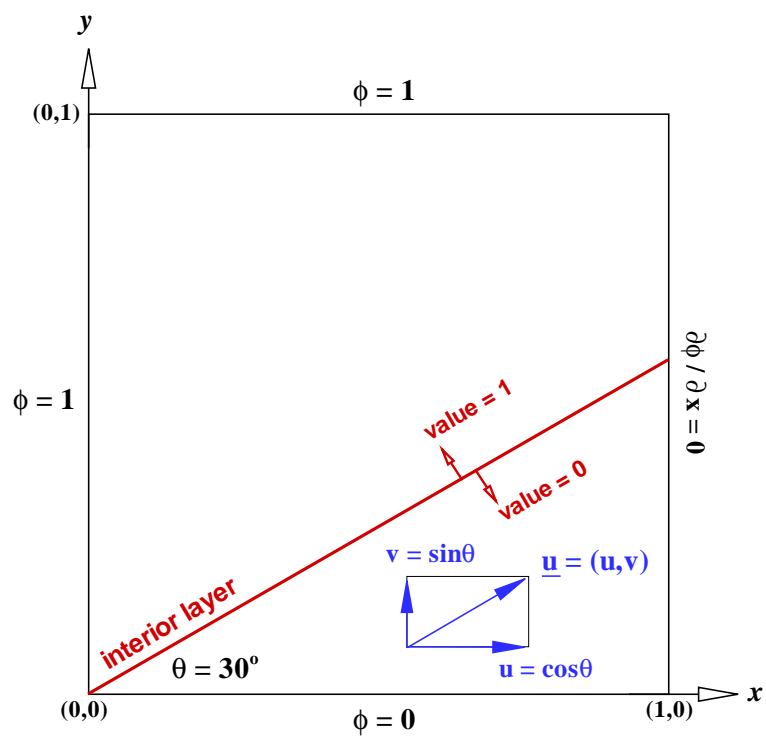
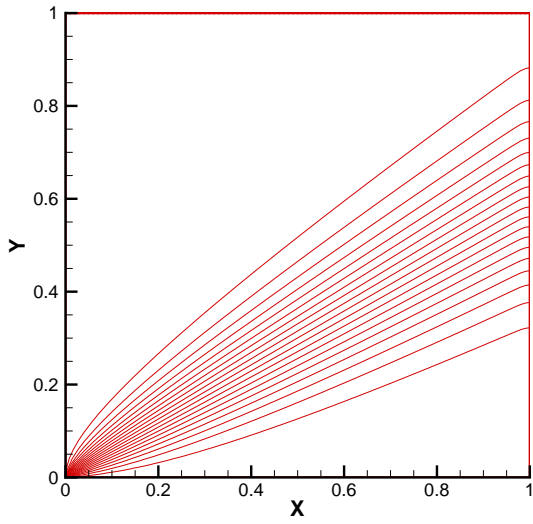
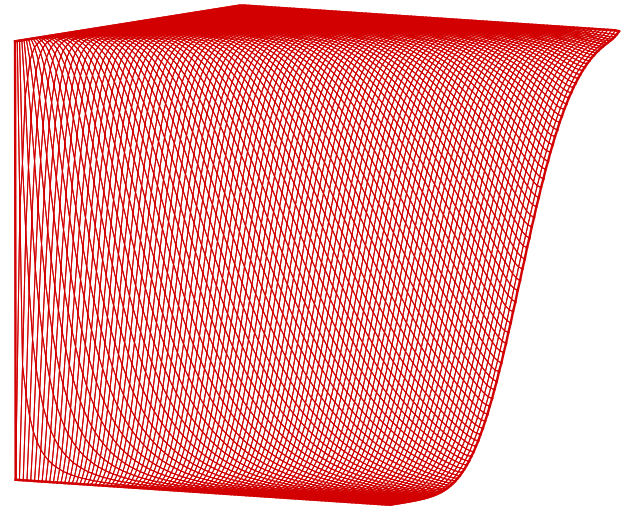


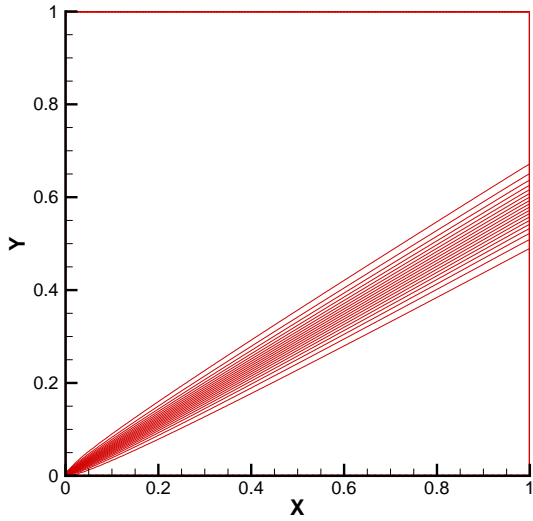
Figure 3.2: Schematic of the skew problem considered in Sec. 3.5.2



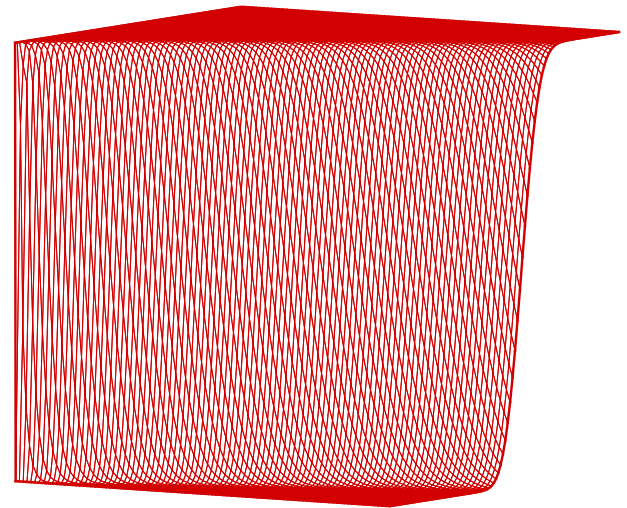
(a)



(b)



(c)



(d)

Figure 3.3: The predicted two- and three-dimensional contours of ϕ for the skew problem considered in Sec. 3.5.2. (a)-(b) $\text{Re} = 100$; (c)-(d) $\text{Re} = 1000$.

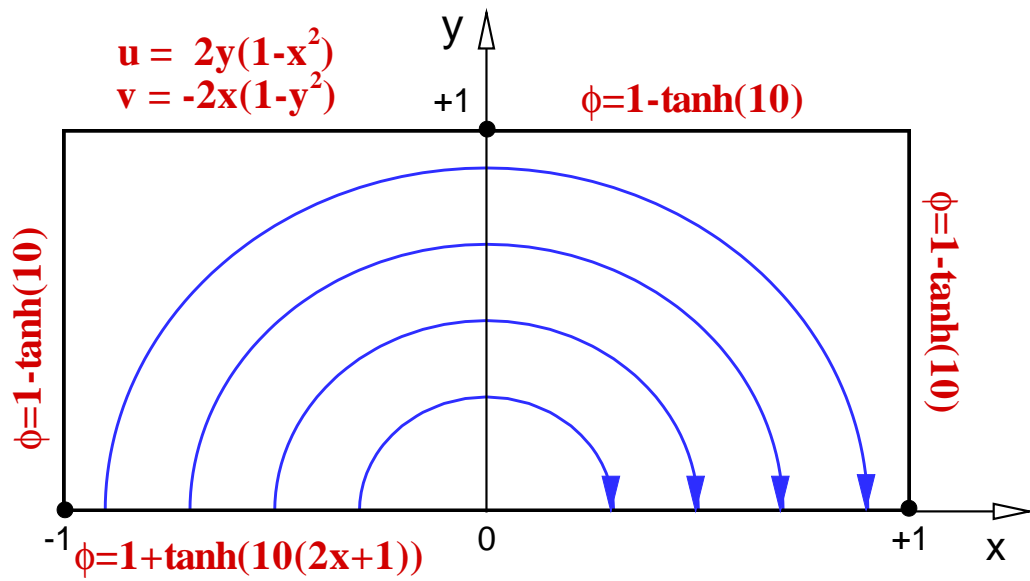


Figure 3.4: Schematic of the Smith & Hutton problem considered in Sec. 3.5.3.

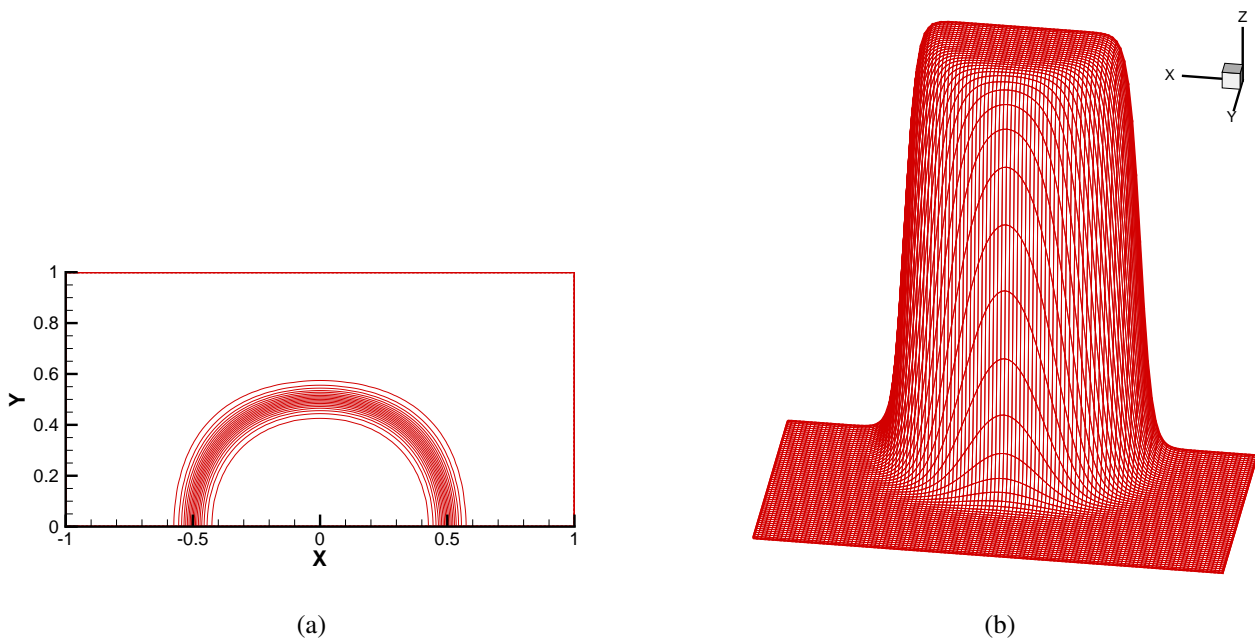
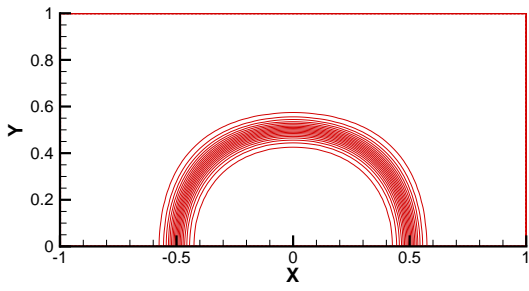
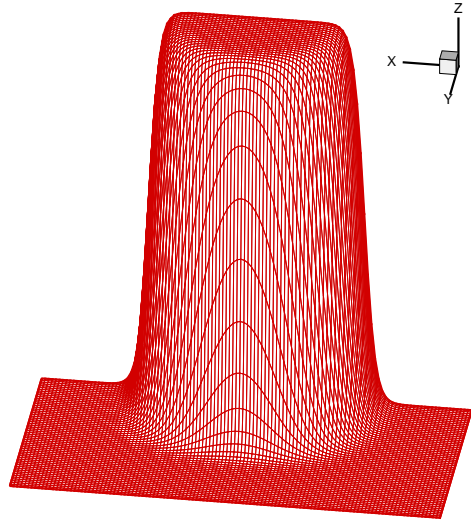


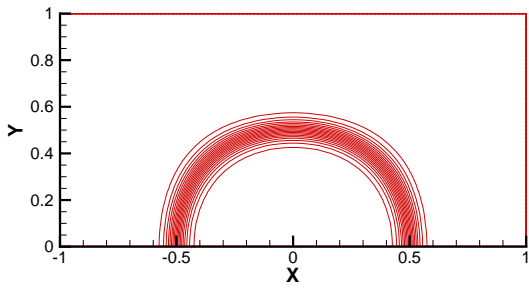
Figure 3.5: The predicted two- and three-dimensional contours of ϕ for the Smith & Hutton problem considered in Sec. 3.5.3. (a)-(b) $Re = 10^8$.



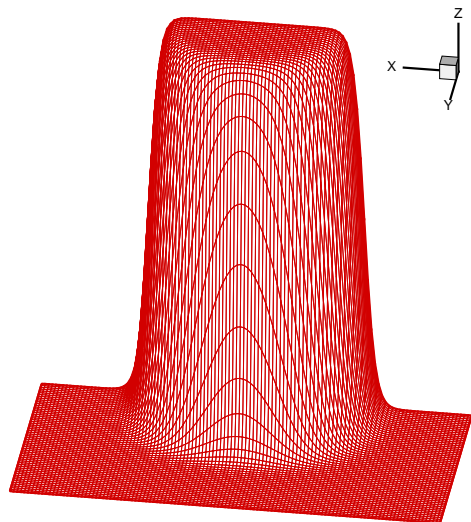
(a)



(b)

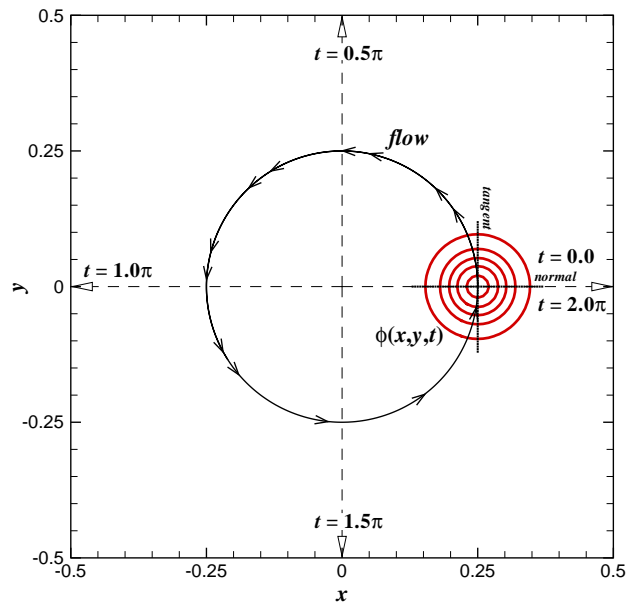


(c)

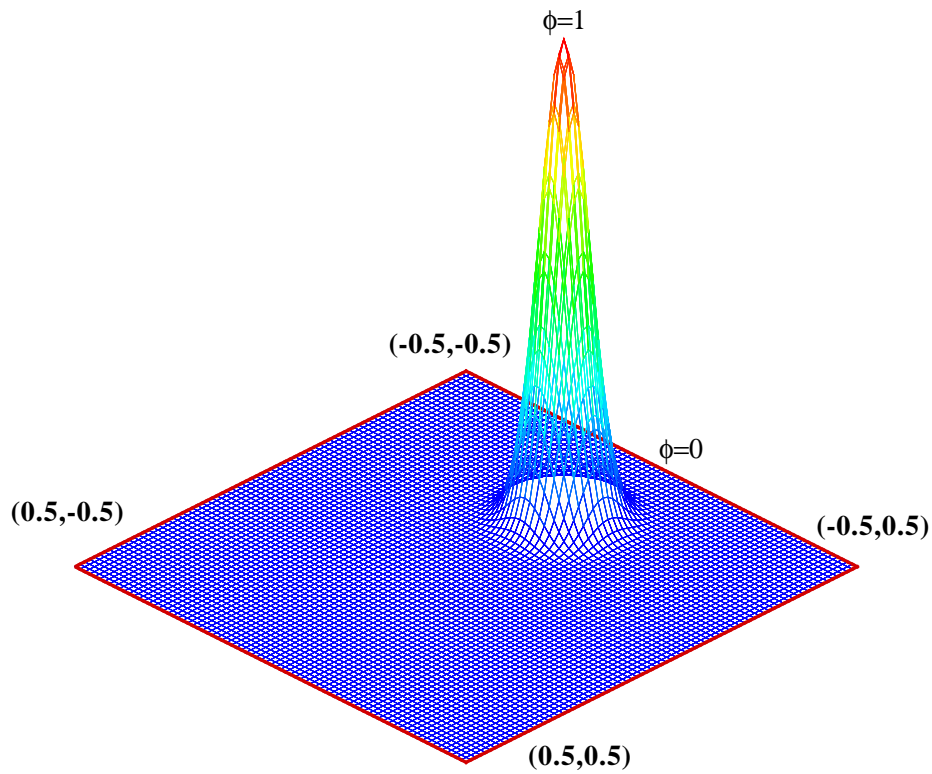


(d)

Figure 3.6: The predicted two- and three-dimensional contours of ϕ for the Smith & Hutton problem considered in Sec. 3.5.3. (a)-(b) $Re=10^8$; (c)-(d) $Re=10^{12}$.



(a)



(b)

Figure 3.7: (a) Schematic of the rotation shaped cone problem considered in Sec. 3.5.4 ;
 (b) Initial condition.

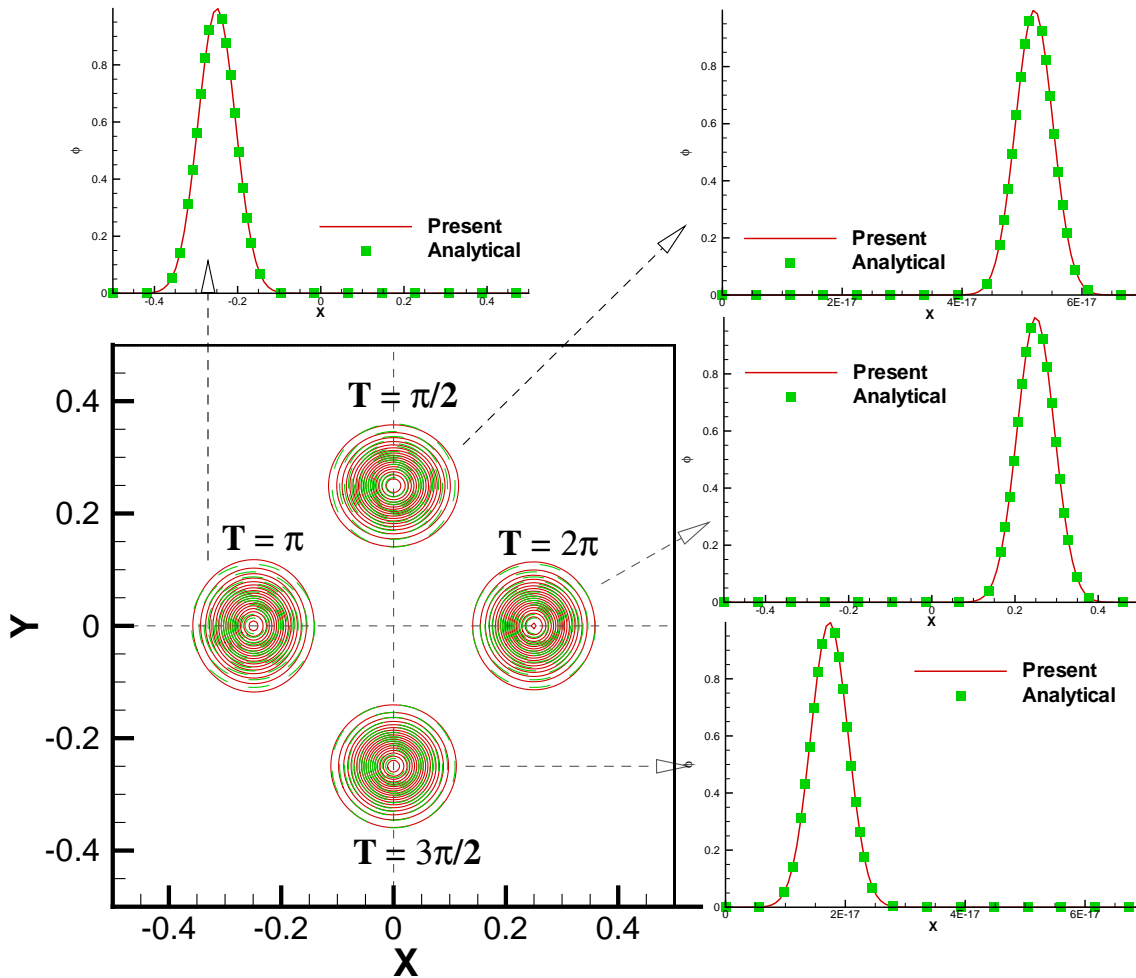
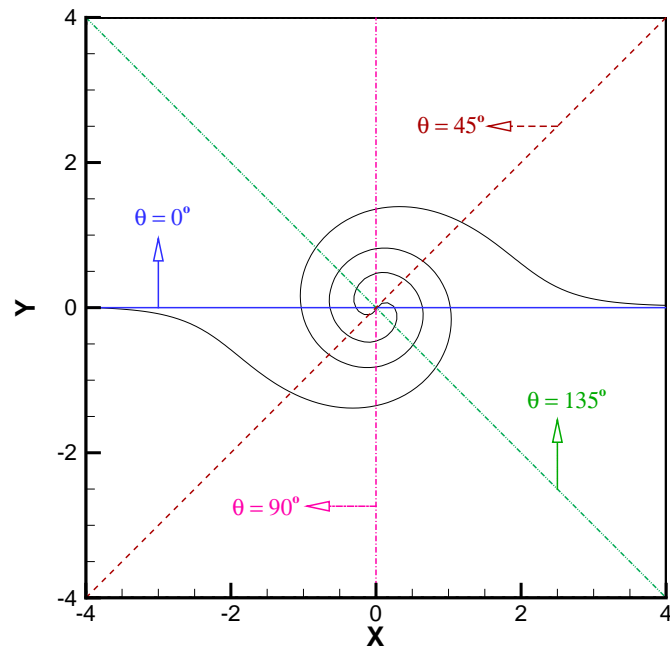
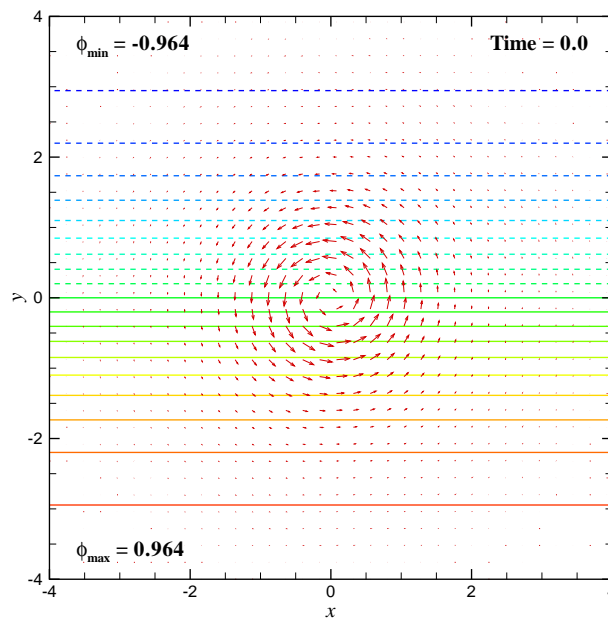


Figure 3.8: The predicted solution contours of ϕ for the rotation shaped cone problem considered in Sec. 3.5.4. (a) $t = \frac{\pi}{2}$; (b) $t = \pi$; (c) $t = \frac{3\pi}{2}$; (d) $t = 2\pi$.

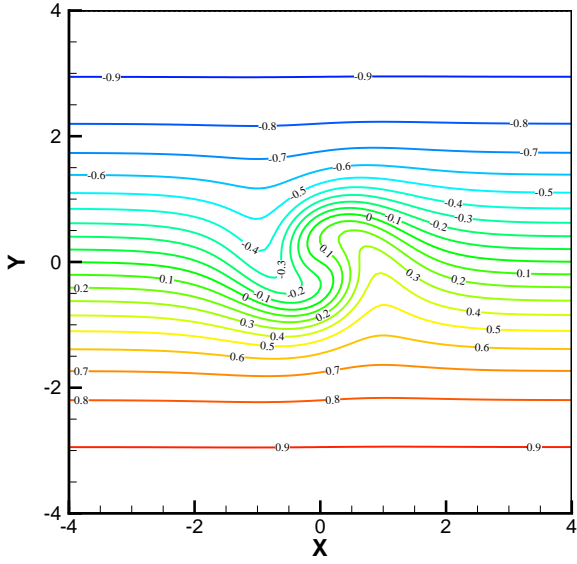


(a)

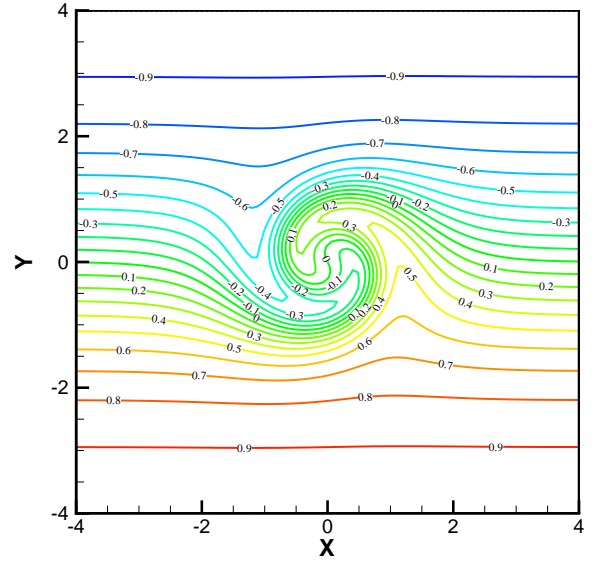


(b)

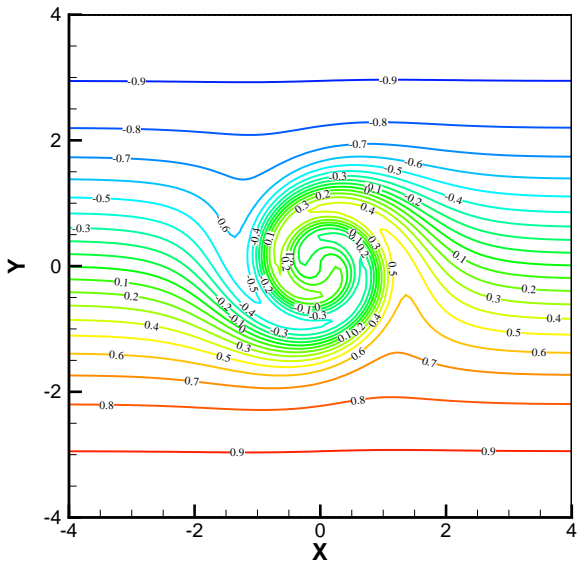
Figure 3.9: (a) Schematic of the mixing warm and cold fluids problem considered in Sec. 3.5.5 ; (b) Initial condition.



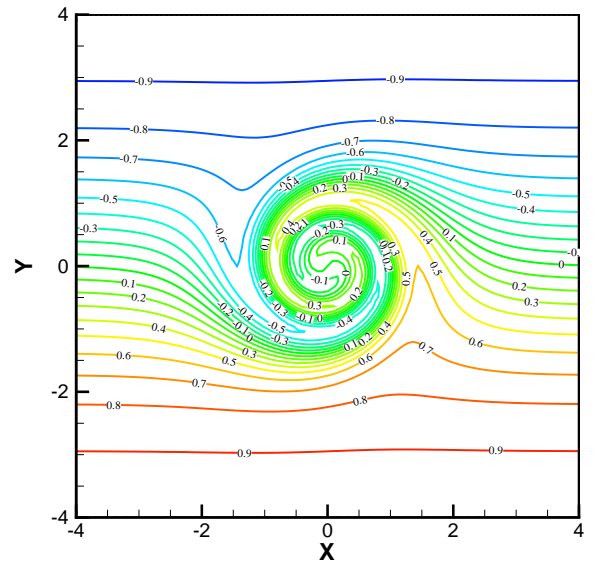
(a)



(b)

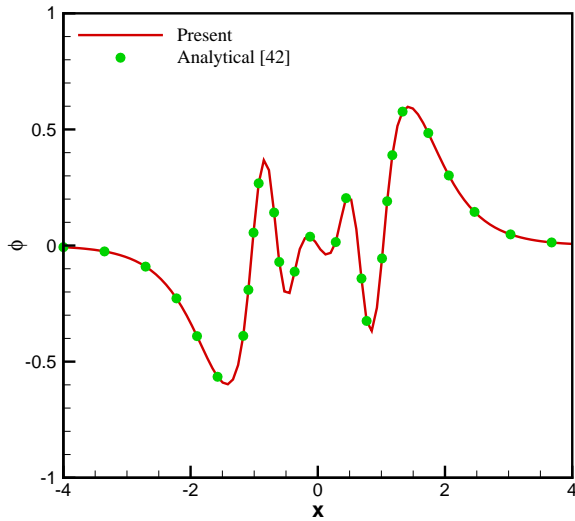


(c)

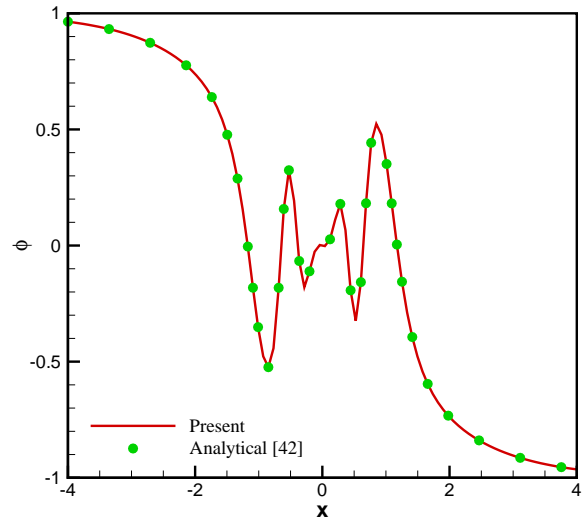


(d)

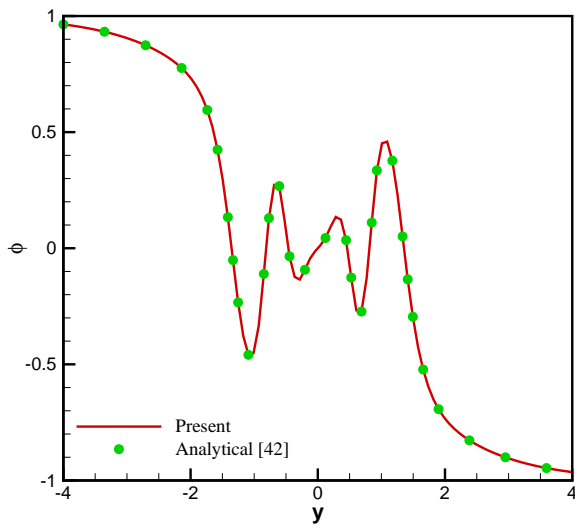
Figure 3.10: The predicted time-evolving temperature contours ϕ for the mixing warm and cold fluids problem considered in Sec. 3.5.5. (a) $t = 1s$; (b) $t = 2s$; (c) $t = 3s$; (d) $t = 4s$.



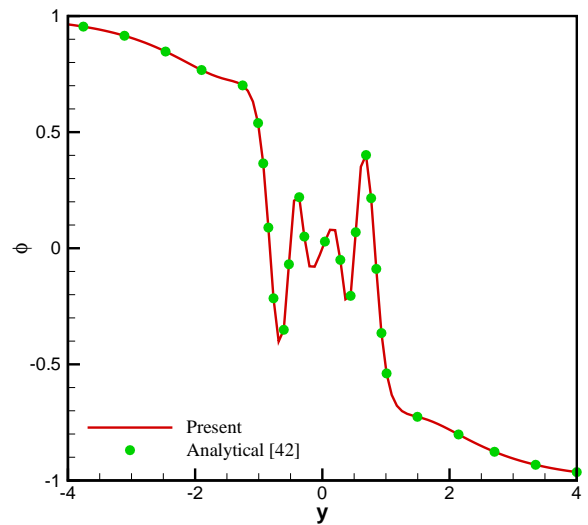
(a)



(b)



(c)



(d)

Figure 3.11: Comparison of predicted solutions with the analytical solutions for the mixing warm and cold fluids problem considered in Sec. 3.5.5, where θ is defined in Fig. 3.9. (a) $\theta = 0^\circ$; (b) $\theta = 45^\circ$; (c) $\theta = 90^\circ$; (d) $\theta = 135^\circ$.



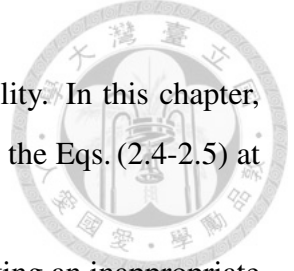
Chapter 4

Finite element model for the incompressible Navier-Stokes equations

The methods for seeking a numerical solution of the incompressible Navier-Stokes equations have been investigated in the past few decades. Considerable interests have been focused on the incompressible viscous flow in order to get a better understanding of the underlying physics in the oceanography, meteorology, hemodynamics, mechanical fluid flow, etc. The purpose of this chapter is to seek the finite element solution of the incompressible Navier-Stokes equations. The short introduction will be given in Section 4.1 and the weak variational formulation of the incompressible Navier-Stokes equations will be introduced in Section 4.2. Next, the difficulty regarding to the incompressibility constrain condition is described in Section 4.3. The interpolation functions for the velocity and the pressure field and finite element formulation are introduced in Section 4.4-4.5. As usual, some analytical verifications and benchmark numerical results are presented in Section 4.6.

4.1 Introduction

It is well known that there are two main potential sources of numerical instability when solving the incompressible Navier-Stokes equations by using the finite element method. One is due to the classical Galerkin treatment of the convection term and it results in spurious node-to-node velocity oscillations. These oscillations become significant as the Reynolds number increases. To overcome this difficulty, the Petrov-Galerkin finite ele-



ment formulation has been employed to enhance convective instability. In this chapter, the developed MWE finite element model will be employed to solve the Eqs. (2.4-2.5) at high Reynolds number.

The other source of numerical instability is observed when adopting an inappropriate combination of the interpolation functions for approximating the velocity and the pressure fields. This numerical instability usually appears as oscillations in the pressure field. The incompressible constraint condition, which makes the pressure field independent of the velocity field, has long been regarded as the primary source of this numerical instability. Taylor and Hood [43] employed the equal-order interpolation functions and found that a smooth velocity solution was usually accompanied by an oscillatory pressure solution. They also found that if the order of interpolation function for pressure is one lower than that for the velocity, non-oscillatory velocity and pressure solutions can be obtained simultaneously. Sani et. al. [44] noticed that the use of equal-order interpolation functions generally results in a singular matrix which produces oscillatory pressure solution.

In Section 2.3, the mixed finite element formulation for solving Eqs. (2.4-2.5) is recommended since it unconditionally satisfies the incompressibility (or divergence-free) constraint condition. However, this is subject to the satisfaction of \mathcal{LBB} (Ladyzhenskage-Babůska-Brezzi) condition [45–47]. The necessity of imposing \mathcal{LBB} condition inhibits an arbitrary combination of the interpolation functions for the velocity and pressure fields. In other words, a pair of interpolation function endowed with the \mathcal{LBB} condition is required to apply to the mixed finite element formulation. For example, for the quadrilateral element with bi-quadratic velocity interpolation function, a discontinuous bi-linear interpolation function is adopted for pressure solutions.

4.2 Weak formulation

Using the standard notations and definitions of the functional spaces defined in Section. 3.3. We denote by \mathcal{S}_u and \mathcal{S}_p the trial functional spaces for the velocity and pressure,



respectively.

$$\begin{aligned}\mathcal{S}_u &= \{\underline{\mathbf{u}} \mid \underline{\mathbf{u}} \in \mathcal{H}^1(\Omega), \underline{\mathbf{u}} = \underline{\mathbf{g}} \text{ on } \partial\Omega = \Gamma\} \\ \mathcal{S}_p &= \{\underline{\mathbf{p}} \mid \underline{\mathbf{p}} \in \mathcal{L}^2(\Omega), \int_{\Omega} \underline{\mathbf{p}} d\Omega = 0\}\end{aligned}$$

In conjunction with \mathcal{S}_u and \mathcal{S}_p , we define the weighting functional spaces for the momentum and continuity equations, denoted by \mathcal{V}_u and \mathcal{V}_p

$$\begin{aligned}\mathcal{V}_u &= \{\underline{\mathbf{w}} \mid \underline{\mathbf{w}} \in \mathcal{H}^1(\Omega), \underline{\mathbf{w}} = 0 \text{ on } \Gamma\} \\ \mathcal{V}_p &= \mathcal{S}_p\end{aligned}$$

Notes that the \mathcal{S}_u and \mathcal{V}_u only differ in the definition of boundary conditions, that is, the weighting functions for the momentum equations vanish on the boundary where the fluid velocity is prescribed. The trial and weighting functional spaces for continuity equation are identical.

Given the above functional spaces, the weak formulation for the incompressible Navier-Stokes equations is derived by multiplying the Eq. (2.4) by $\underline{\mathbf{w}} \in \mathcal{V}_u$ and Eq. (2.5) by $\underline{\mathbf{q}} \in \mathcal{V}_p$ and integrating the resulting equations over the domain Ω .

$$\begin{aligned}\int_{\Omega} \frac{\partial \underline{\mathbf{u}}}{\partial t} \underline{\mathbf{w}} d\Omega + \int_{\Omega} (\underline{\mathbf{u}} \cdot \nabla \underline{\mathbf{u}}) \underline{\mathbf{w}} d\Omega - \frac{1}{Re} \int_{\Omega} \Delta \underline{\mathbf{u}} \underline{\mathbf{w}} d\Omega + \int_{\Omega} \nabla \underline{\mathbf{p}} \cdot \underline{\mathbf{w}} d\Omega \\ = \int_{\Omega} \underline{\mathbf{f}} \underline{\mathbf{w}} d\Omega\end{aligned}\quad (4.1)$$

$$\int_{\Omega} (\nabla \cdot \underline{\mathbf{u}}) \underline{\mathbf{q}} d\Omega = 0\quad (4.2)$$

By applying the Green's theorem [10] on the third and fourth terms in Eq. (4.1), one can obtain

$$\begin{aligned}\int_{\Omega} \frac{\partial \underline{\mathbf{u}}}{\partial t} \underline{\mathbf{w}} d\Omega + \int_{\Omega} (\underline{\mathbf{u}} \cdot \nabla \underline{\mathbf{u}}) \underline{\mathbf{w}} d\Omega + \frac{1}{Re} \int_{\Omega} \nabla \underline{\mathbf{u}} : \nabla \underline{\mathbf{w}} d\Omega - \int_{\Omega} \underline{\mathbf{p}} (\nabla \cdot \underline{\mathbf{w}}) d\Omega \\ + \frac{1}{Re} \int_{\Gamma} (\nabla \underline{\mathbf{u}}) \underline{\mathbf{w}} \cdot \underline{\mathbf{n}} d\Gamma - \int_{\Gamma} \underline{\mathbf{p}} \underline{\mathbf{w}} \cdot \underline{\mathbf{n}} d\Gamma = \int_{\Omega} \underline{\mathbf{f}} \cdot \underline{\mathbf{w}} d\Omega\end{aligned}\quad (4.3)$$

$$\int_{\Omega} (\nabla \cdot \underline{\mathbf{u}}) \underline{\mathbf{q}} d\Omega = 0\quad (4.4)$$

Note that the integral term over the boundary in Eq. (4.3) vanishes. The resulting weak variational formulation for the incompressible Navier-Stokes equations is stated as fol-



lows : Find $(\underline{\mathbf{u}}, \underline{\mathbf{p}}) \in \mathcal{S}_u \times \mathcal{S}_p$ such that

$$B[(\underline{\mathbf{u}}, \underline{\mathbf{p}}), (\underline{\mathbf{w}}, \underline{\mathbf{q}})] = L(\underline{\mathbf{w}}) \quad \forall (\underline{\mathbf{w}}, \underline{\mathbf{q}}) \in \mathcal{V}_u \times \mathcal{V}_p \quad (4.5)$$

where the bi-linear form $B(\cdot, \cdot)$ and linear form $L(\cdot)$ are, respectively, defined below

$$\begin{aligned} B[(\underline{\mathbf{u}}, \underline{\mathbf{p}}), (\underline{\mathbf{w}}, \underline{\mathbf{q}})] &= \left(\frac{\partial \underline{\mathbf{u}}}{\partial t}, \underline{\mathbf{w}}\right) + (\underline{\mathbf{u}} \cdot \nabla \underline{\mathbf{u}}, \underline{\mathbf{w}}) + \frac{1}{Re} (\nabla \underline{\mathbf{u}}, \nabla \underline{\mathbf{w}}) \\ &\quad - (\underline{\mathbf{p}}, \nabla \cdot \underline{\mathbf{w}}) - (\nabla \cdot \underline{\mathbf{u}}, \underline{\mathbf{q}}) \end{aligned} \quad (4.6)$$

$$L(\underline{\mathbf{w}}) = (\underline{\mathbf{f}}, \underline{\mathbf{w}}) \quad (4.7)$$

Let $\mathcal{S}_u^h \subseteq \mathcal{S}_u$, $\mathcal{S}_p^h \subseteq \mathcal{S}_p$, $\mathcal{V}_u^h \subseteq \mathcal{V}_u$ and $\mathcal{V}_p^h \subseteq \mathcal{V}_p$ be the finite-dimensional subspaces for trial and weighting functions. The finite element solutions $(\underline{\mathbf{u}}^h, \underline{\mathbf{p}}^h)$ are sought from the following statement : Find $(\underline{\mathbf{u}}^h, \underline{\mathbf{p}}^h) \in \mathcal{S}_u^h \times \mathcal{S}_p^h$ such that

$$\begin{aligned} B[(\underline{\mathbf{u}}^h, \underline{\mathbf{p}}^h), (\underline{\mathbf{w}}^h, \underline{\mathbf{q}}^h)] &= \left(\frac{\partial \underline{\mathbf{u}}^h}{\partial t}, \underline{\mathbf{w}}^h\right) + (\underline{\mathbf{u}}^h \cdot \nabla \underline{\mathbf{u}}^h, \underline{\mathbf{w}}^h) + \frac{1}{Re} (\nabla \underline{\mathbf{u}}^h, \nabla \underline{\mathbf{w}}^h) \\ &\quad - (\underline{\mathbf{p}}^h, \nabla \cdot \underline{\mathbf{w}}^h) - (\nabla \cdot \underline{\mathbf{u}}^h, \underline{\mathbf{q}}^h) \quad \forall (\underline{\mathbf{w}}^h, \underline{\mathbf{q}}^h) \in (\mathcal{V}_u^h, \mathcal{V}_p^h) \end{aligned} \quad (4.8)$$

$$L(\underline{\mathbf{w}}^h) = (\underline{\mathbf{f}}, \underline{\mathbf{w}}^h) \quad (4.9)$$

4.3 Incompressibility constraint condition

When simulating the incompressible fluid flow, the difficulty in satisfying the incompressibility (or divergence-free) constraint condition Eq. (2.5) has been known to result in numerical instability. Numerical study of the incompressible fluid flows is also of theoretical importance to satisfy the incompressibility constraint for velocity. When considering the incompressible Navier-Stokes equations, pressure serves as the Lagrangian multiplier rather than as the thermodynamics property in compressible flow. As a result, the continuity equation serves as a constraint condition for the velocity field. Maintenance of this constraint condition demands that the pressure field adjust itself instantaneously to the velocity field.

As mentioned already, the pair of interpolation functions must satisfy the $\mathcal{LB}\mathcal{B}$ condition in order to produce the smooth velocity and pressure solutions simultaneously. This condition arises from the mathematical realization: There exists a positive constant β



which is independent of mesh size, such that

$$\inf_{\mathbf{q} \in \mathcal{V}_p} \sup_{\mathbf{w} \in \mathcal{V}_u} \left(\frac{(\mathbf{q}, \nabla \cdot \mathbf{w})}{\|\nabla \mathbf{w}\|_{H^1} \|\mathbf{q}\|_{L^2}} \right) \geq \beta \quad (4.10)$$

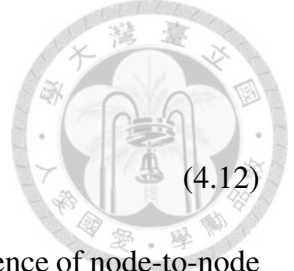
The $\mathcal{LB}\mathcal{B}$ condition is the result of the fact that Eq. (4.10) is an example of the stability condition found in Brezzi's theory for the mixed finite element formulation, which, in turn, is essential when applying the Babuška theory. Ladyzhenska also proved the analogous condition for the continuous case. Note that the $\mathcal{LB}\mathcal{B}$ condition is independent of the nonlinearity of Navier-Stokes equations, it is also satisfied for the Stokes equations. Some methods for the non-primitive variable formulation remove the necessity of using $\mathcal{LB}\mathcal{B}$ condition, but the continuity equation must be modified to avoid the use of the incompressibility constraint.

4.4 Interpolation function

At the first glance, the primitive variables (\mathbf{u}, p) in Eqs. (4.3)-(4.4) seem to be independently approximated through the interpolation functions. However, for ensuring stability, the pair of interpolation functions must satisfy the $\mathcal{LB}\mathcal{B}$ condition. A typical example is the bi-quadratic/bi-linear element pair, proposed by Bercovier and Pironneau [48], satisfies the $\mathcal{LB}\mathcal{B}$ condition and will be employed in two-dimensional finite element calculations. In this regard, the velocity field is approximated by the continuous bi-quadratic interpolation functions N_j ($j = 1 \sim 9$), written in terms of the natural coordinate $-1 \leq \xi \leq 1$ and $-1 \leq \eta \leq 1$.

$$N_j = \begin{cases} \frac{1}{4}(1 + \bar{\xi}_j)(1 + \bar{\eta}_j)(\bar{\xi} + \bar{\eta} - 1) & , j = 1, 3, 5, 7 \\ \frac{1}{2}(1 - \xi^2)(1 + \bar{\eta}) & , j = 2, 6 \\ \frac{1}{2}(1 + \bar{\xi})(1 - \eta^2) & , j = 4, 8 \\ (1 - \xi^2)(1 - \eta^2) & , j = 9 \end{cases} \quad (4.11)$$

where $\bar{\xi} = \xi\xi_j$, $\bar{\eta} = \eta\eta_j$ denote the normalized coordinates of the j -th node. To satisfy the Eq. (4.10), the pressure is approximated by the discontinuous bi-linear interpolation



functions M_l given by

$$M_l = \frac{1}{4}(1 + \xi\xi_l)(1 + \eta\eta_l), \quad l = 1 \sim 4 \quad (4.12)$$

This element pair is very attractive due to its easy coding and the absence of node-to-node pressure oscillations.

4.5 Finite element formulation

When simulating an incompressible fluid flow, we demand the divergence-free discrete velocities be attainable. This can be achieved by employing the mixed coupling approach instead of segregated uncoupling approach so that the mass and momentum conservation can be simultaneously attained. In Eqs. (4.8)-(4.9), the finite element solutions for the velocity and the pressure can be expressed as follows :

$$\underline{\mathbf{u}}^h(\xi, \eta) = \sum_{j=1}^9 N_j(\xi, \eta) \underline{\mathbf{u}}_j \quad (4.13)$$

$$\underline{\mathbf{p}}^h(\xi, \eta) = \sum_{l=1}^4 M_l(\xi, \eta) \underline{\mathbf{p}}_l \quad (4.14)$$

where $\underline{\mathbf{u}}_j$ and $\underline{\mathbf{p}}_l$ are the unknowns values at the nodal points. By substituting Eqs. (4.13)-(4.14) into the weak formulation Eqs. (4.8)-(4.9), we can derive the following matrix equations along with the bi-linear weighting function \mathbf{q}^h for the continuity equation.

$$\underline{\mathbf{T}} \begin{pmatrix} \frac{\partial \underline{\mathbf{u}}_j}{\partial t} \\ \frac{\partial \underline{\mathbf{v}}_j}{\partial t} \\ 0 \end{pmatrix} + \underline{\mathbf{A}} \begin{pmatrix} \underline{\mathbf{u}}_j \\ \underline{\mathbf{v}}_j \\ \underline{\mathbf{p}}_l \end{pmatrix} = \underline{\mathbf{b}} \quad (4.15)$$

where the mass matrix $\underline{\underline{\mathbf{T}}}$ and fluid matrix $\underline{\underline{\mathbf{A}}}$ are defined as follows

$$\underline{\underline{\mathbf{T}}} = \int_{\Omega} \begin{pmatrix} N_i N_j & 0 & 0 \\ 0 & N_i N_j & 0 \\ 0 & 0 & 0 \end{pmatrix} d\Omega, \quad \underline{\underline{\mathbf{A}}} = \int_{\Omega} \begin{pmatrix} C_{ij} & 0 & -M_l \frac{\partial N_i}{\partial x} \\ 0 & C_{ij} & -M_l \frac{\partial N_i}{\partial y} \\ -M_l \frac{\partial N_j}{\partial x} & -M_l \frac{\partial N_j}{\partial y} & 0 \end{pmatrix} d\Omega$$

$$C_{ij} = (N_i + B_i)(N_m \tilde{u}_m) \frac{\partial N_j}{\partial x_k} + \frac{1}{Re} \frac{\partial N_i}{\partial x_k} \frac{\partial N_j}{\partial x_k}$$

In practice, it is customary to set the value of \tilde{u}_m as a constant in order to linearize the momentum equations. The tilde \sim in C_{ij} denotes velocities obtained at the previous nonlinear iteration step. The right hand side vector $\underline{\mathbf{b}}$ includes the source and surface integral terms. When the natural (traction-free) boundary conditions are imposed, $\underline{\mathbf{b}}$ can be derived as

$$\underline{\mathbf{b}} = - \int_{\Gamma_{\text{out}}} \begin{pmatrix} p_l \mathbf{n}_x - \frac{1}{Re} \frac{\partial u_j}{\partial \underline{\mathbf{n}}} \\ p_l \mathbf{n}_y - \frac{1}{Re} \frac{\partial v_j}{\partial \underline{\mathbf{n}}} \\ 0 \end{pmatrix} d\Gamma$$

where $(\mathbf{n}_x, \mathbf{n}_y)$ denotes the outward normal vector, on which the pressure is imposed. For the temporal derivatives, the 2nd-order backward Euler implicit scheme is employed

$$\frac{\partial \mathbf{u}}{\partial t} = \frac{1}{2\Delta t} \left(3\mathbf{u}^{(n)} - 4\mathbf{u}^{(n-1)} + \mathbf{u}^{(n-2)} \right) \quad (4.16)$$

$$\frac{\partial \mathbf{v}}{\partial t} = \frac{1}{2\Delta t} \left(3\mathbf{v}^{(n)} - 4\mathbf{v}^{(n-1)} + \mathbf{v}^{(n-2)} \right) \quad (4.17)$$

For the first time step, the 1st-order backward Euler scheme is employed. Note that there is no restriction about the time step since the backward Euler scheme is implicit. By

substituting the Eqs. (4.14)-(4.15) into Eq. (4.13) yields

$$\underline{\mathbf{T}} \begin{pmatrix} \frac{1}{2\Delta t} \left(3\mathbf{u}^{(n)} - 4\mathbf{u}^{(n-1)} + \mathbf{u}^{(n-2)} \right) \\ \frac{1}{2\Delta t} \left(3\mathbf{v}^{(n)} - 4\mathbf{v}^{(n-1)} + \mathbf{v}^{(n-2)} \right) \\ 0 \end{pmatrix} + \underline{\mathbf{A}} \begin{pmatrix} \mathbf{u}_j^{(n)} \\ \mathbf{v}_j^{(n)} \\ \mathbf{p}_l^{(n)} \end{pmatrix} = \underline{\mathbf{b}}, \quad (4.18)$$

Rearranging the Eq. (4.18), the final finite element formulation is read as

$$\left(\frac{3}{2\Delta t} \underline{\mathbf{T}} + \underline{\mathbf{A}} \right) \begin{pmatrix} \mathbf{u}_j^{(n)} \\ \mathbf{v}_j^{(n)} \\ \mathbf{p}_l^{(n)} \end{pmatrix} = \frac{1}{2\Delta t} \underline{\mathbf{T}} \begin{pmatrix} 4\mathbf{u}_j^{(n-1)} - \mathbf{u}_j^{(n-2)} \\ 4\mathbf{v}_j^{(n-1)} - \mathbf{v}_j^{(n-2)} \\ 0 \end{pmatrix} + \underline{\mathbf{b}} \quad (4.19)$$

The Gaussian elimination-based direct Frontal solver developed by Irons [49, 50] is used to solve the above matrix equations. It is a variant of Gaussian elimination that automatically avoids a large number of operations involving zero components. The direct Frontal solver begins by assembling the matrix for each element. This is followed by incorporating all the elementary matrices into the global matrix system. As soon as all the contributions from each element to a particular node have been assembled, the corresponding variables associated with this node can then be eliminated immediately. This process continues until all the elementary matrices have been assembled and the elimination procedure is completed. Calculation of the solution is followed by performing backward substitution. The Frontal solver is considered as an effective solver for two-dimensional problem. Unfortunately, it is infeasible in three-dimensional problem due to the largely increased memory demands and operation costs. Consequently, the iterative solver turns out to be the legitimate alternative in three-dimensional problem.

4.6 Verification study

For verifying the proposed MWE finite element model for solving the incompressible Navier-Stokes equations, two problems amenable to the analytical solutions are investi-



gated in the domain of $[0, 1] \times [0, 1]$

4.6.1 Steady-state analytical verification problem

The steady-state Navier-Stokes equations with the analytical velocities prescribed at the boundary [51] will be solved at $Re = 100$

$$\begin{aligned}u(\mathbf{x}, \mathbf{y}) &= \frac{-2(1 + \mathbf{y})}{(1 + \mathbf{x})^2 + (1 + \mathbf{y})^2} \\v(\mathbf{x}, \mathbf{y}) &= \frac{2(1 + \mathbf{x})}{(1 + \mathbf{x})^2 + (1 + \mathbf{y})^2}\end{aligned}$$

According to the above solutions, the corresponding analytical pressure solution can be derived as

$$p(\mathbf{x}, \mathbf{y}) = -\frac{2}{(1 + \mathbf{x})^2 + (1 + \mathbf{y})^2}$$

From Table 4.1, good agreement with the analytical solutions is seen. In addition, good rate of convergence obtained from the computed L_2 -error norms at $21^2, 41^2, 61^2$ and 81^2 nodal points is obtained.

4.6.2 Transient analytical verification problem

The transient Navier-Stokes equations amenable to the following analytical solutions are also solved for the verification purpose

$$\begin{aligned}u(\mathbf{x}, \mathbf{y}, t) &= -\cos(\pi\mathbf{x}) \sin(\pi\mathbf{y}) \exp(-2\pi^2 t/Re) \\v(\mathbf{x}, \mathbf{y}, t) &= \sin(\pi\mathbf{x}) \cos(\pi\mathbf{y}) \exp(-2\pi^2 t/Re) \\p(\mathbf{x}, \mathbf{y}, t) &= -\frac{1}{4}(\cos(2\pi\mathbf{x}) + \cos(2\pi\mathbf{y})) \exp(-2\pi^2 t/Re)\end{aligned}$$

All the calculations are carried out at different nodal sizes $21^2, 41^2, 61^2$ and 81^2 and time step $\Delta t = 0.1$. The computed L_2 error norms at $t = 4$ and $t = 8$ tabulated in Tables. 4.1-4.2 show good agreements with the analytical solutions. The applicability of the proposed finite element scheme to solve the steady or transient incompressible Navier-Stokes equations is, therefore, confirmed.



4.7 Numerical results

4.7.1 Lid-driven cavity problem

The lid-driven cavity problem schematic in Fig. 4.1 is regarded as the standard benchmark problem because of its simple geometry, easy boundary condition implementation and its embedded rich physical flow phenomena. The boundary velocities are zero everywhere except along the lid-plane at $y = 1$. All the calculations with 129^2 and 151^2 nodal points are carried out in the unit domain. The residual reduction profiles for all the investigated cases are plotted in Fig. 4.2. The predicted streamfunction contours are plotted in Fig. 4.3. For the sake of comparison, the predicted mid-section horizontal $u(0.5, y)$ and vertical $v(x, 0.5)$ velocity profiles plotted in Fig. 4.3 show good agreement with the steady-state benchmark solutions of Ghia [52] and Erturk [53]. In addition, the velocity profile near the boundary layer at $Re = 10000$ are also compared well with the reference solutions of Coupez [54]. The proposed model is also applied to solve the transient problem case. For the case at $Re = 400$, 129^2 nodal points and $\Delta t = 0.1$, the residual reduction profile is plotted in Fig. 4.5. The predicted velocity profiles shown in Fig. 4.6 and Fig. 4.7 also show good agreement with the reference solutions of Pontaza [55] and Dailey [56].

For the sake of completeness, the predicted eddy centers at T, BL1, BR1, BL2 and BR2, shown in Fig. 4.1, are also compared well with the reference data that are summarized in Tables 4.4-4.5. The applicability of the proposed finite element model to predict the high Reynolds number incompressible flows is, therefore, confirmed.

4.7.2 Backward facing step flow problem

The second benchmark problem deals with the channel flow with a backward-facing step. This well known problem has been extensively employed in code validation due to its simple geometry and the presence of interesting flow phenomena such as flow separation, flow reattachment and multiple recirculations. The problem is schematically shown in Fig. 4.8. The ratio of the height of the backward-facing step, h , to the height of the downstream channel, H , is chosen to be $h : H = 1 : 2$. The previous study of Wang and Sheu [57]

revealed that as $L/h > 32$, the flow solution becomes trustable since the simulated traction force is zero. The downstream channel length is, thus, chosen as 20 in this study. Two different cases schematic in Fig. 4.8 are investigated in the Reynolds number range of $100 \leq Re \leq 800$. The fully-developed flow profile $\mathbf{u}(0.5 \leq \mathbf{y} \leq 1) = 24(1 - \mathbf{y})(\mathbf{y} - 0.5)$ is prescribed at the inlet boundary. No-slip condition is imposed at every solid wall and the traction-free condition is imposed at the outlet boundary. The residual reduction profiles at $Re = 800$ are plotted in Fig. 4.9. For the sake of comparison, the predicted velocity profiles $\mathbf{u}(3, \mathbf{y})$ and $\mathbf{u}(7, \mathbf{y})$ at $Re=800$ plotted in Fig. 4.10 are compared with the reference solutions of Gartling [58] and Erturk [59] with good agreement.

We denote x_1 as the computed reattachment length of the recirculation region downstream of the step. The x_2 and x_3 are denoted as the separation and reattachment lengths of the upper stream of the step, The comparison of x_1 in the range of $100 \leq Re \leq 800$ with the reference data of Erturk [59] in Fig. 4.11 shows that the predicted x_1 reattachment lengths are shorter for the cases without inlet channel. In addition, the comparisons of the predicted x_2 separation and x_3 reattachment lengths with other reference solutions [58, 60–64, 66] are also tabulated in Table. 4.6.

4.7.3 Natural convection problem

Two-dimensional benchmark natural convection problem in an unit domain ($0 \leq \mathbf{x}, \mathbf{y} \leq 1$) schematically shown in Fig. 4.12 is investigated by the following equations

$$\begin{aligned} \nabla \cdot \underline{\mathbf{u}} &= 0 \\ \frac{\partial \underline{\mathbf{u}}}{\partial t} + \underline{\mathbf{u}} \cdot \nabla \underline{\mathbf{u}} &= -\nabla \mathbf{p} + Pr \nabla^2 \underline{\mathbf{u}} + RaPr \mathbf{T}_y \\ \frac{\partial T}{\partial t} + \underline{\mathbf{u}} \cdot \nabla T &= \nabla^2 T \end{aligned} \quad (4.20)$$

Here T represents the temperature field, Pr and Ra denote the Prandtl and Rayleigh number, respectively. The boundary velocities at all walls are no-slip, while the temperatures along the left hot wall are $T = 1$ and right cold wall are set to be $T = 0$. Along the horizontal walls $\mathbf{y} = 0$ and $\mathbf{y} = 1$, the temperatures are assumed to be adiabatic ($\frac{\partial T}{\partial \mathbf{y}} = 0$). The cases are investigated at $Ra = 10^3, 10^4, 10^5, 10^6, 10^7$ and 10^8 and $Pr = 0.71$. Uniform

distribution of 129×129 nodal points is employed for the cases with $Ra = 10^3 \sim 10^7$ and 257×257 for the case with $Ra = 10^8$. The residual reduction profiles for the cases $Ra = 10^7$ and 10^8 are plotted in Fig 4.13. The predicted streamfunction and temperature contours at $Ra = 10^7$ and 10^8 are plotted in Fig. 4.14. From Table 4.7, the predicted values of streamfunction at (0.5,0.5) at different Ra numbers are compared well with other reference solutions given in [67–69]. In addition, the predicted velocity v and temperature T profiles plotted in Figs. 4.15-4.17 also show good agreements with the reference solutions of Najafi [67].

For the sake of completeness, we can calculate the local Nusselt number Nu_x and its average value \overline{Nu} , which describe the heat transfer characteristic across the cavity, is defined as

$$Nu_x = \int_0^1 Q(x, y) dy \quad (4.21)$$

$$\overline{Nu} = \int_0^1 Nu_x dx \quad (4.22)$$

where $Q(x, y) = uT - \frac{\partial T}{\partial x}$ denotes the local heat flux at any point in the cavity. The Simpson rule is used to evaluate the Eqs. (4.21)-(4.22). One can clearly see from Fig. 4.17 and Table. 4.8 that the predicted local Nusselt number at hot wall and \overline{Nu} are compared well with other results given in [68–71].



grid sizes	u -velocity		v -velocity		p -pressure	
	L_2 -norms	R.O.C.	L_2 -norms	R.O.C.	L_2 -norms	R.O.C.
21x21	1.125E-4	–	1.194E-4	–	1.575E-3	–
41x41	1.289E-5	3.126	1.343E-5	3.152	4.018E-4	1.971
61x61	3.060E-6	3.547	3.156E-6	3.572	1.790E-4	1.994
81x81	1.044E-6	3.738	1.074E-6	3.747	1.006E-4	2.003

Table 4.1: The predicted L_2 error norms for the steady-state analytical verification problem considered in Sec. 4.6.1.



grid sizes	u -velocity		v -velocity		p -pressure	
	L_2 -norms	R.O.C.	L_2 -norms	R.O.C.	L_2 -norms	R.O.C.
21x21	1.060E-2	–	1.329E-2	–	1.045E-2	–
41x41	1.908E-3	2.474	2.154E-3	2.625	5.073E-3	1.470
61x61	6.375E-4	2.704	7.107E-4	2.735	2.869E-3	1.406
81x81	2.888E-4	2.752	3.119E-4	2.863	2.010E-3	1.237

Table 4.2: The predicted L_2 error norms at $t = 4$ for the transient analytical verification problem considered in Sec. 4.6.2.

grid sizes	u -velocity		v -velocity		p -pressure	
	L_2 -norms	R.O.C.	L_2 -norms	R.O.C.	L_2 -norms	R.O.C.
21x21	9.554E-3	–	1.197E-2	–	1.213E-2	–
41x41	1.677E-3	2.510	1.902E-3	2.654	4.365E-3	1.475
61x61	5.559E-4	2.723	6.182E-4	2.774	2.444E-3	1.430
81x81	2.501E-4	2.776	2.697E-4	2.883	1.710E-3	1.241

Table 4.3: The predicted L_2 error norms at $t = 8$ for the transient analytical verification problem considered in Sec. 4.6.2.



Symbol	Authors	Re			
		1,000	5,000	7,500	10,000
Primary	Ghia et. al. [52]	-0.1179	-0.1189	-0.1199	-0.1197
	Present	-0.1184	-0.1244	-0.1252	-0.1254
	Location (x,y)	(0.5313,0.5625)	(0.5117,0.5352)	(0.5117,0.5322)	(0.5117,0.5333)
T	Ghia et. al. [52]	—	1.4564×10^{-3}	2.0462×10^{-3}	2.4201×10^{-3}
	Present	—	1.2370×10^{-3}	1.8700×10^{-3}	2.3205×10^{-3}
	Location (x,y)	—	(0.0625,0.9102)	(0.0664,0.9141)	(0.0703,0.9141)
BL1	Ghia et. al. [52]	2.3112×10^{-4}	1.3611×10^{-3}	1.4670×10^{-3}	1.5128×10^{-3}
	Present	2.2360×10^{-4}	1.3130×10^{-3}	1.4577×10^{-3}	1.5265×10^{-3}
	Location (x,y)	(0.0859,0.0781)	(0.0703,0.1367)	(0.0645,0.1504)	(0.0586,0.1641)
BR1	Ghia et. al. [52]	1.7510×10^{-3}	3.0835×10^{-3}	3.2848×10^{-3}	3.4183×10^{-3}
	Present	1.6892×10^{-3}	2.9855×10^{-3}	3.0928×10^{-3}	3.0280×10^{-3}
	Location (x,y)	(0.8594,0.1094)	(0.8086,0.0742)	(0.7813,0.0625)	(0.7656,0.0586)
BL2	Ghia et. al. [52]	—	-7.0886×10^{-8}	-1.8316×10^{-7}	-7.7565×10^{-7}
	Present	—	-5.9553×10^{-8}	-2.2905×10^{-7}	-9.2063×10^{-7}
	Location (x,y)	—	(0.0117,0.0078)	(0.0117,0.0117)	(0.0156,0.0195)
BR2	Ghia et. al. [52]	-9.3192×10^{-8}	-1.4322×10^{-6}	-3.2814×10^{-5}	-1.3132×10^{-4}
	Present	-7.6148×10^{-8}	-1.3834×10^{-6}	-2.6210×10^{-5}	-1.2016×10^{-4}
	Location (x,y)	(0.9922,0.0078)	(0.9805,0.0195)	(0.9492,0.0430)	(0.9336,0.0625)
BR3	Ghia et. al. [52]	—	—	1.5811×10^{-9}	5.6683×10^{-9}
	Present	—	—	1.1146×10^{-9}	3.3595×10^{-9}
	Location (x,y)	—	—	(0.9961,0.0039)	(0.9961,0.0039)

Table 4.4: Comparison of the predicted streamfunction solutions with the results of Ghia [52] for the lid-driven cavity problem considered in Sec. 4.7.1.



Symbol	Authors	Re			
		1,000	5,000	7,500	10,000
Primary	Erturk et. al. [53]	-0.1187	-0.1212	-0.1209	-0.1204
	Present	-0.1184	-0.1245	-0.1252	-0.1254
	Location (x,y)	(0.5300,0.5650)	(0.5150,0.5350)	(0.5133,0.5317)	(0.5117,0.5300)
T	Erturk et. al. [53]	—	1.4416×10^{-3}	2.1119×10^{-3}	2.5870×10^{-3}
	Present	—	1.2379×10^{-3}	1.8714×10^{-3}	2.3252×10^{-3}
	Location (x,y)	—	(0.0633,0.9100)	(0.0667,0.9133)	(0.0717,0.9117)
BL1	Erturk et. al. [53]	2.3266×10^{-4}	1.3639×10^{-3}	1.5171×10^{-3}	1.5930×10^{-3}
	Present	2.2399×10^{-4}	1.3162×10^{-3}	1.4572×10^{-3}	1.5250×10^{-3}
	Location (x,y)	(0.0833,0.0783)	(0.0733,0.1367)	(0.0650,0.1517)	(0.0583,0.1633)
BR1	Erturk et. al. [53]	1.7281×10^{-3}	3.0835×10^{-3}	3.2102×10^{-3}	3.1758×10^{-3}
	Present	1.6892×10^{-3}	2.9855×10^{-3}	3.1088×10^{-3}	3.0659×10^{-3}
	Location (x,y)	(0.8633,0.1117)	(0.8086,0.0742)	(0.7900,0.0650)	(0.7767,0.0600)
BL2	Erturk et. al. [53]	-8.4221×10^{-8}	-7.2080×10^{-8}	-2.0202×10^{-7}	-1.0151×10^{-6}
	Present	-2.1484×10^{-8}	-9.7635×10^{-8}	-2.2905×10^{-7}	-9.1216×10^{-7}
	Location (x,y)	(0.0050,0.0050)	(0.0083,0.0083)	(0.0117,0.0117)	(0.0167,0.0200)
BR2	Erturk et. al. [53]	-5.4962×10^{-8}	-1.4010×10^{-6}	-3.0998×10^{-5}	-1.3397×10^{-4}
	Present	-7.2006×10^{-8}	-1.4068×10^{-6}	-2.7026×10^{-5}	-1.2018×10^{-4}
	Location (x,y)	(0.9917,0.0067)	(0.9783,0.0183)	(0.9517,0.0417)	(0.9350,0.0667)
BR3	Erturk et. al. [53]	—	2.1562×10^{-10}	1.4409×10^{-9}	5.1934×10^{-9}
	Present	—	7.9339×10^{-10}	3.7625×10^{-9}	4.0978×10^{-9}
	Location (x,y)	—	(0.9983,0.0017)	(0.9967,0.0033)	(0.9967,0.0050)

Table 4.5: Comparison of the predicted streamfunction solutions with the results of Erturk [53] for the lid-driven cavity problem considered in Sec. 4.7.1.



Authors		Lower eddy reattachment x_1/h	Upper eddy separation x_2/h	Upper eddy reattachment x_3/h
No inlet channel	Gartling [58]	6.10	4.85	10.48
	Gresho et. al. [60]	6.08	4.84	10.46
	Gresho et. al. [60]	6.10	4.86	10.49
	Sani and Gresho [61]	6.22	5.09	10.25
	Barton [62]	6.02	4.81	10.48
	Keskar and Lyn [63]	6.10	4.85	10.48
	Present	5.97	5.02	10.29
With inlet channel ($l = 25$)	Sheu and Hsu [65]	5.73	4.65	10.02
	Wan, Patnaik and Wei [64]	5.02	—	—
	Abide (Parabolic inlet) [66]	5.90	—	—
	Abide (uniform inlet) [66]	5.06	—	—
	Present (parabolic inlet)	5.80	4.93	10.03
Present (uniform inlet)	5.80	4.93	10.03	

Table 4.6: Comparison of the reattachment and separation lengths for the backward-facing step flow problem investigated at $Re = 800$.



Ra	10^3	10^4	10^5	10^6	10^7	10^8
Najafi and Enjilela [67]	1.184	5.083	9.076	16.12	29.38	51.15
Kalita et. al [68]	1.174	5.071	9.111	16.32	—	—
Ramawamy, Jue and Akin [68]	1.170	5.099	9.217	16.68	29.34	—
Dennis and Hudson [68]	1.175	5.074	9.113	—	—	—
Kalita, Dalal and Dass [68]	1.175	5.080	9.123	16.42	29.38	—
Sheu and Chiu [69]	1.174	5.070	9.103	16.35	29.43	—
Present work	1.174	5.066	9.125	16.40	29.34	52.27

Table 4.7: Comparison of the predicted streamfunction at (0.5,0.5) for the natural convection problem considered in Sec. 4.7.3.

Ra	10^3	10^4	10^5	10^6	10^7	10^8
Chenoweth and Paolucci [68]	1.118	2.244	4.520	8.822	16.82	—
De Vahl Davis [68]	1.118	2.243	4.519	8.800	—	—
Ball and Kuo [68]	1.118	2.248	4.528	8.824	16.52	—
Ho and Lin [68]	1.118	2.248	4.528	8.824	16.52	—
Kalita, Dalal and Dass [68]	1.118	2.245	4.522	8.829	16.52	—
Sheu and Chiu [69]	1.118	2.241	4.528	8.820	16.70	—
Dixit and Babu [70]	1.121	2.286	4.546	8.652	—	—
Wan, Patnaik and Wei [71]	1.073	2.155	4.352	8.632	13.86	23.67
Present work	1.118	2.242	4.515	8.810	16.48	23.22

Table 4.8: Comparison of the predicted averaged Nusselt numbers \overline{Nu} for the natural convection problem considered in Sec. 4.7.3.

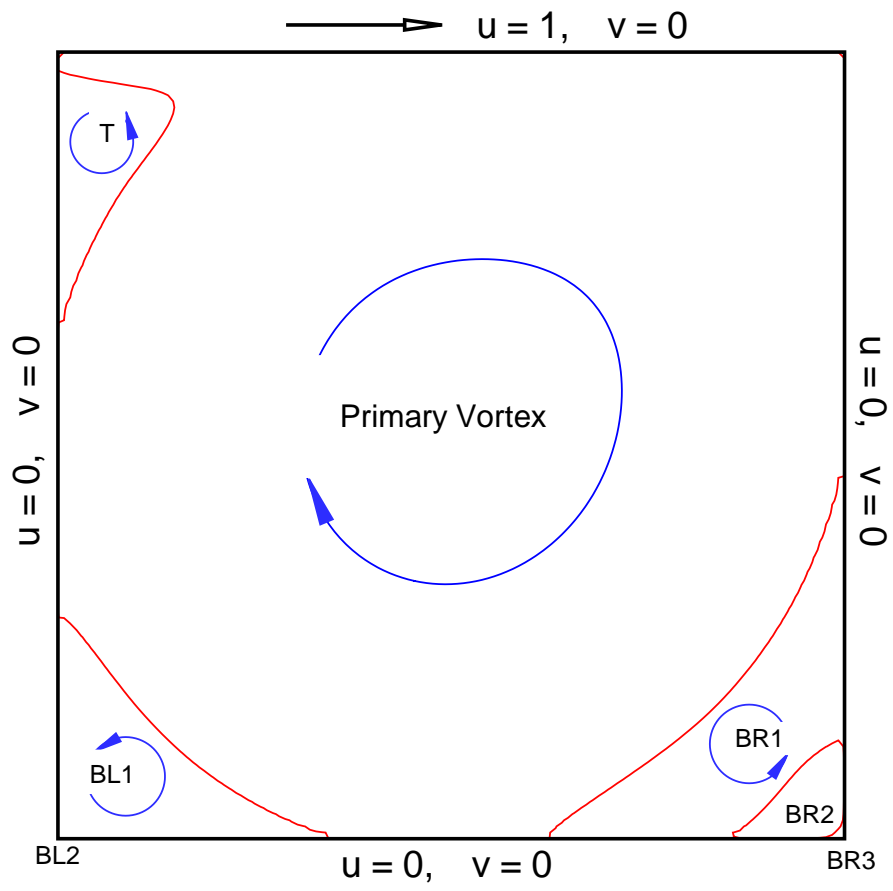
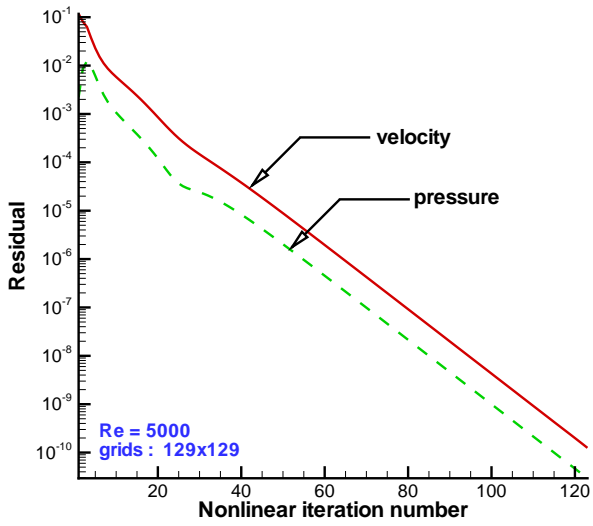
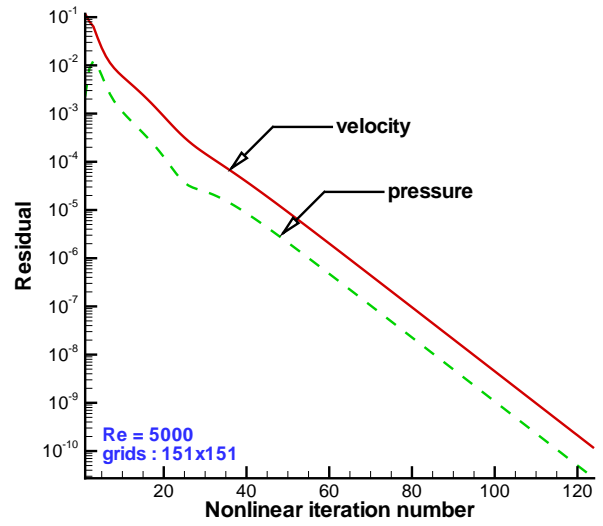


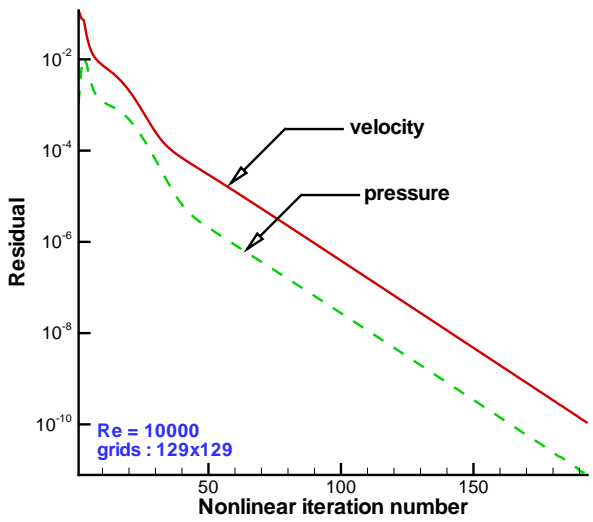
Figure 4.1: Schematic of the lid-driven cavity problem considered in Sec. 4.7.1



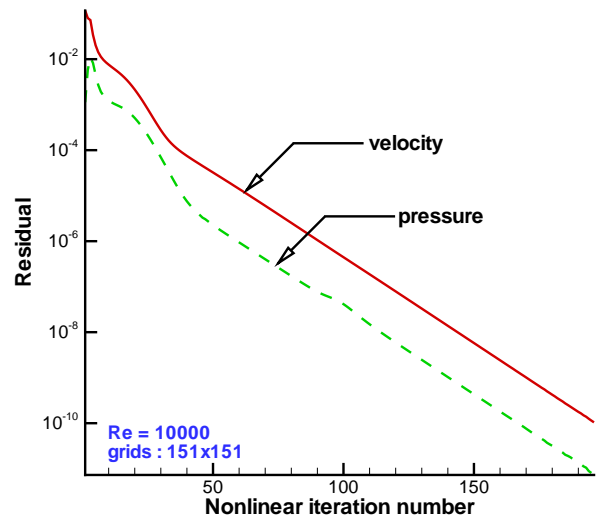
(a)



(b)

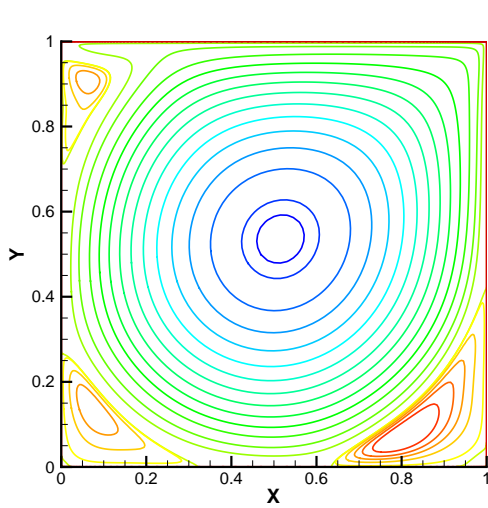


(c)

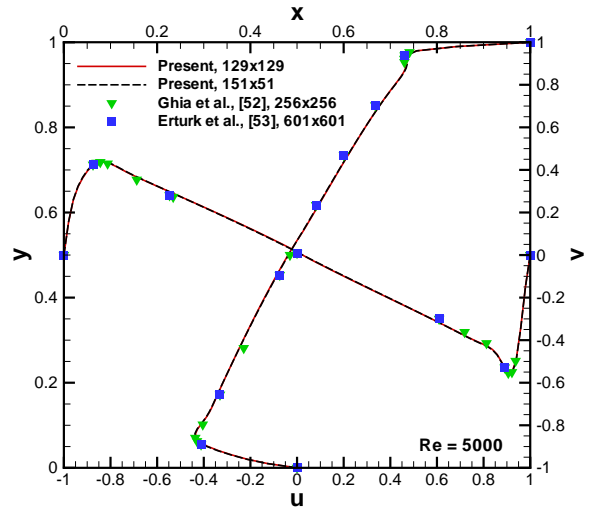


(d)

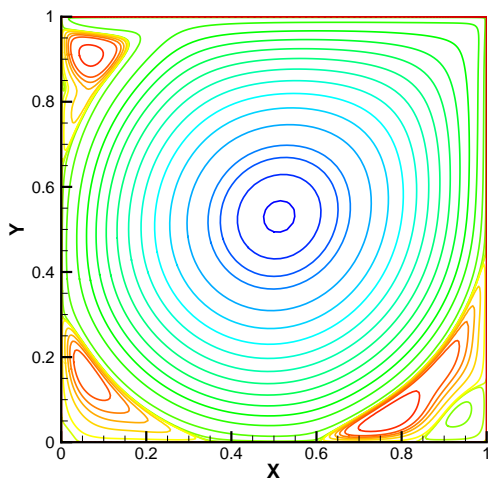
Figure 4.2: Residual reduction plots for the steady-state lid-driven cavity problem considered in Sec. 4.7.1. (a)-(b) $Re = 5000$; (c)-(d) $Re = 10000$.



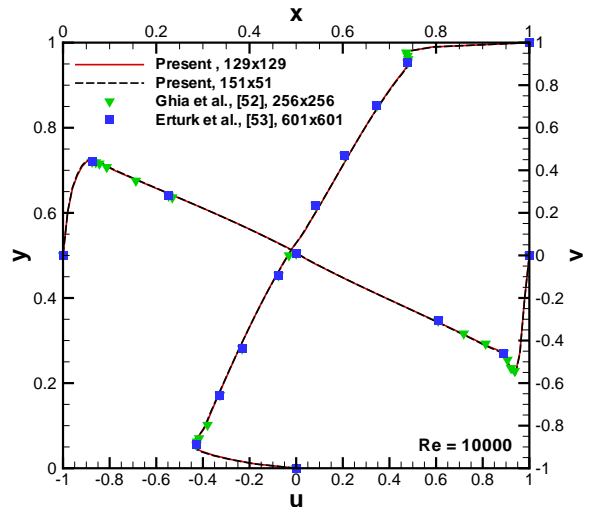
(a)



(b)



(c)



(d)

Figure 4.3: Comparison of the streamfunction contours and velocity profiles $u(0.5, y)$ and $v(x, 0.5)$ for the steady-state lid-driven cavity problem considered in Sec. 4.7.1. (a)-(b) $Re = 5000$; (c)-(d) $Re = 10000$.

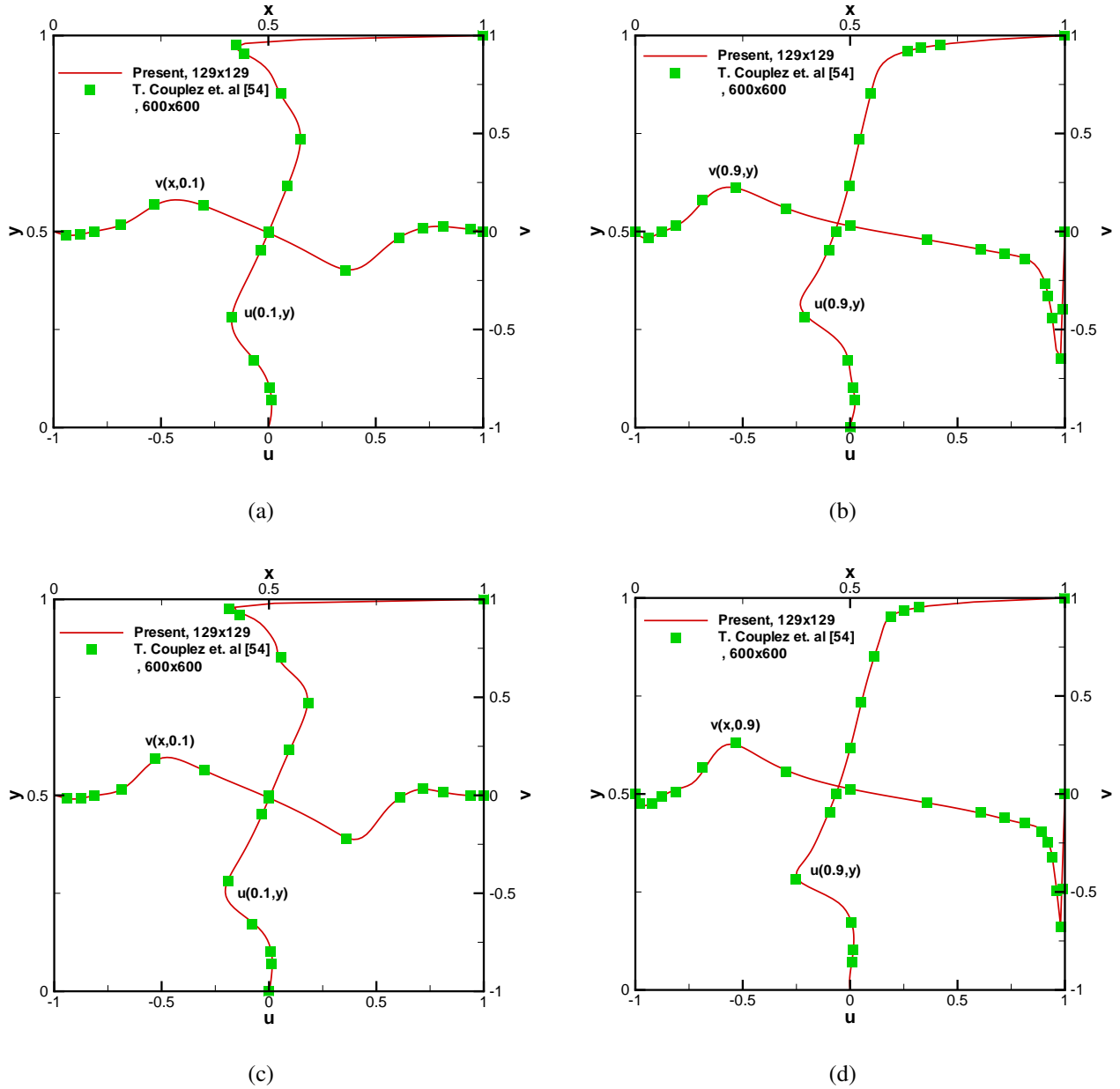


Figure 4.4: Comparisons of the predicted velocity profiles $u(0.1, y)$, $u(0.9, y)$, $v(x, 0.1)$ and $v(x, 0.9)$ for the steady-state lid-driven cavity problem considered in Sec. 4.7.1. (a)-(b) $Re = 5000$; (c)-(d) $Re = 10000$

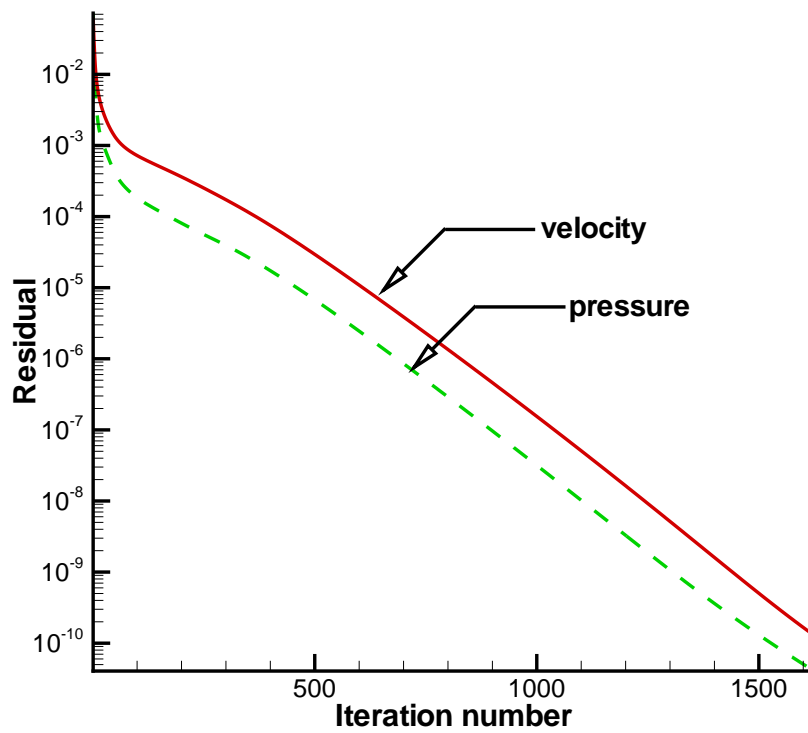


Figure 4.5: Residual reduction plot investigated at $Re = 400$ for the transient lid-driven cavity flow problem considered in Sec. 4.7.1.

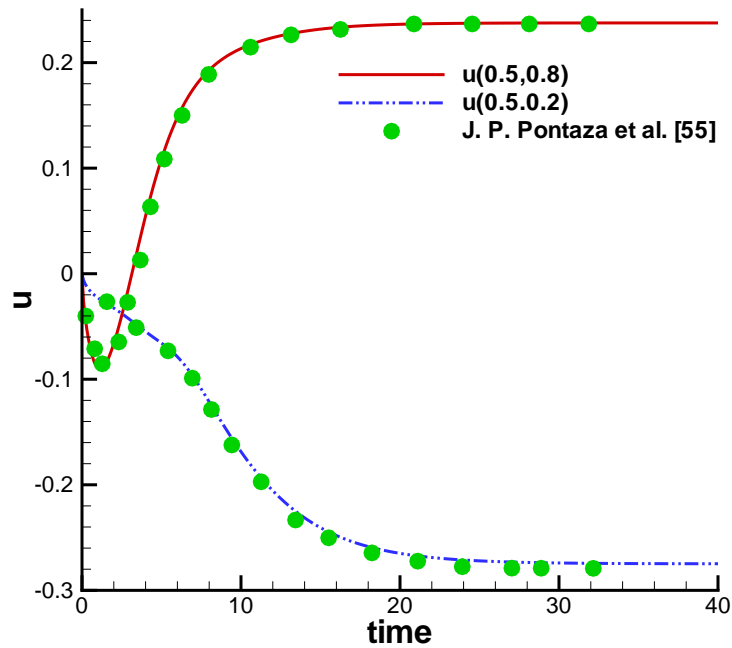


Figure 4.6: Comparison of the predicted velocity profiles $u(0.5,0.2)$ and $u(0.5,0.8)$ investigated at $Re = 400$ with the results of Pontaza's [55] for the transient lid-driven cavity problem considered in Sec. 4.7.1.

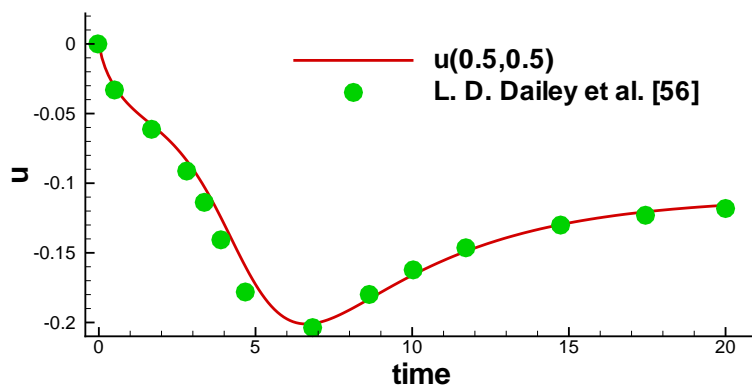
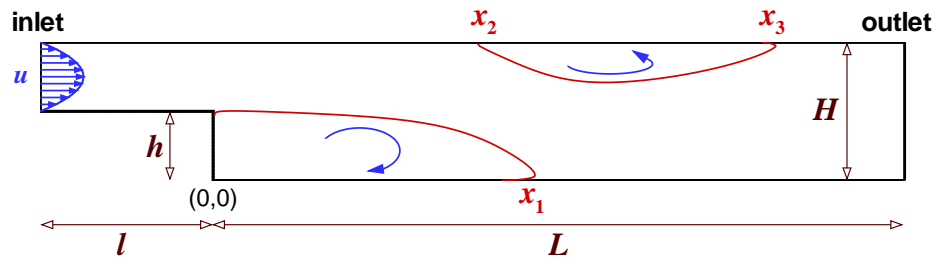
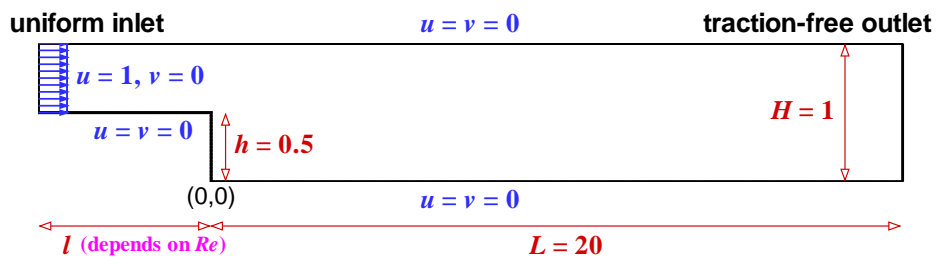


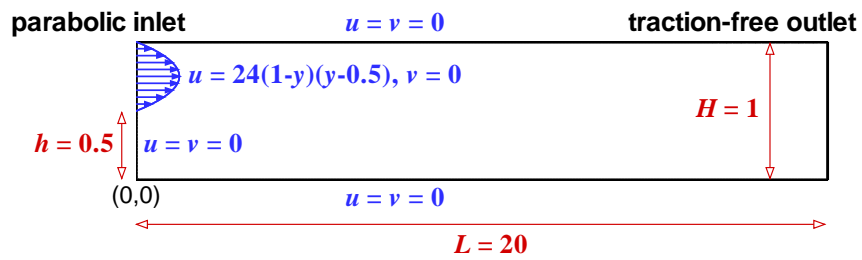
Figure 4.7: Comparison of the predicted velocity profile $u(0.5,0.5)$ investigated at $Re = 400$ with the results of Dailey's [56] for the transient lid-driven cavity considered in Sec. 4.7.1.



(a)

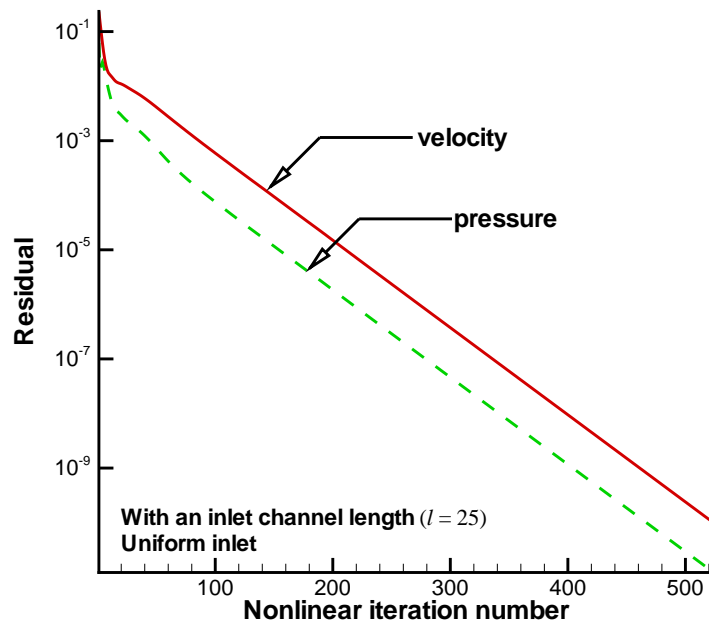


(b)

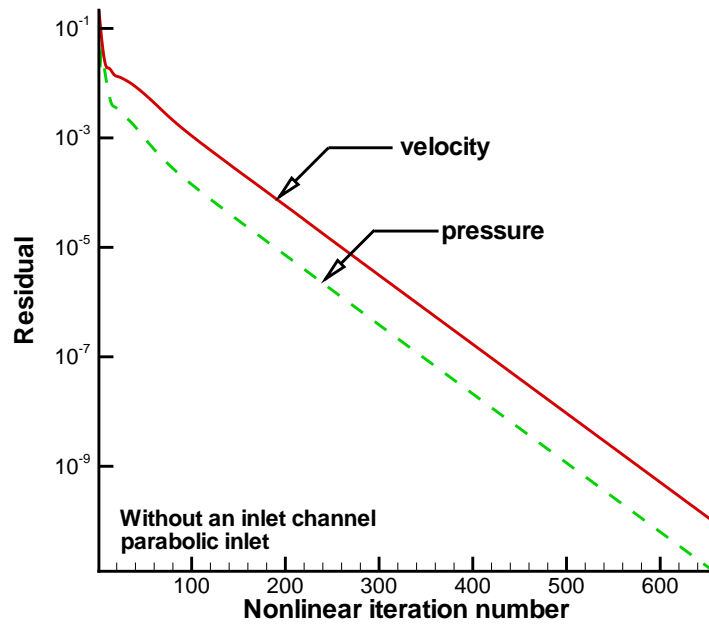


(c)

Figure 4.8: Schematic of the backward-facing step flow problem considered in Sec. 4.7.2 (a) Problem geometry and recirculation regions ; (b) Uniform inlet flow profile with an inlet flow channel ; (c) Fully-developed inlet flow profile without an inlet flow channel.



(a)



(b)

Figure 4.9: Residual reduction plots for the steady-state backward-facing step flow problem at $Re=800$ considered in Sec. 4.7.2 (a) With an inlet channel ; (b) Without an inlet channel

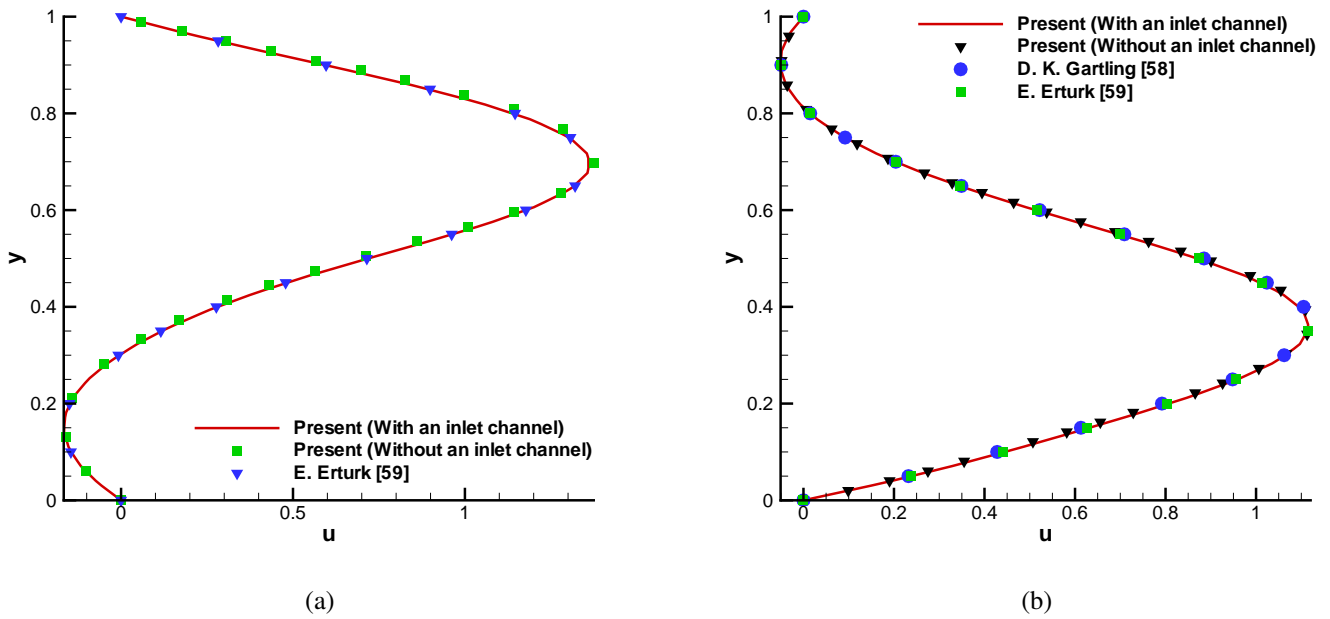


Figure 4.10: Comparison of the predicted u -velocity profiles investigated at $Re = 800$ with the results of Gartling [58] and Erturk [59] for the steady-state backward-facing step flow problem considered in Sec. 4.7.2 (a) $x = 3$; (b) $x = 7$.

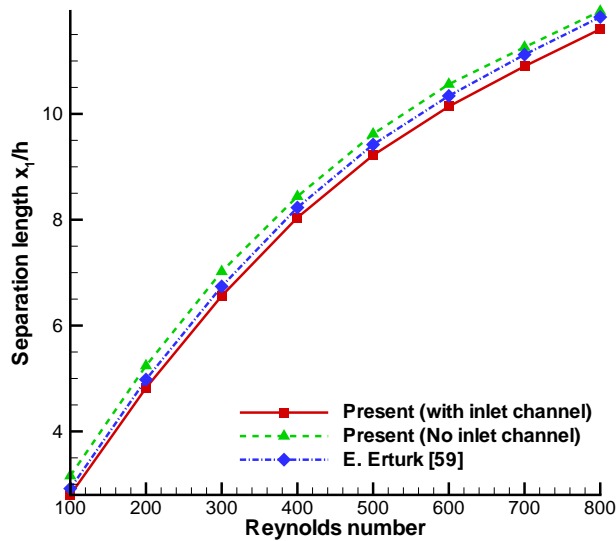


Figure 4.11: Comparison of the reattachment lengths x_1/h with the results of Erturk [59] for the backward-facing step flow problem considered in section 4.7.2.

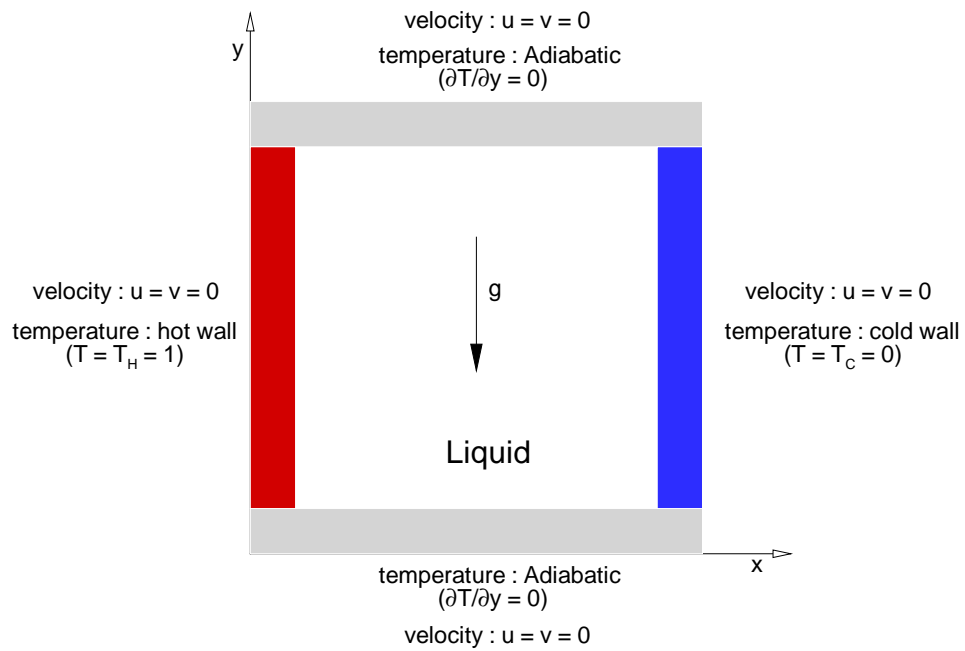
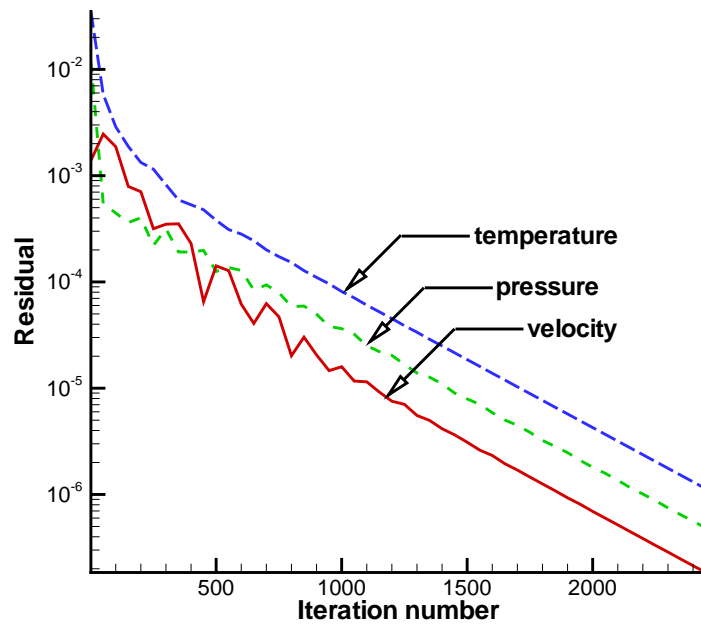
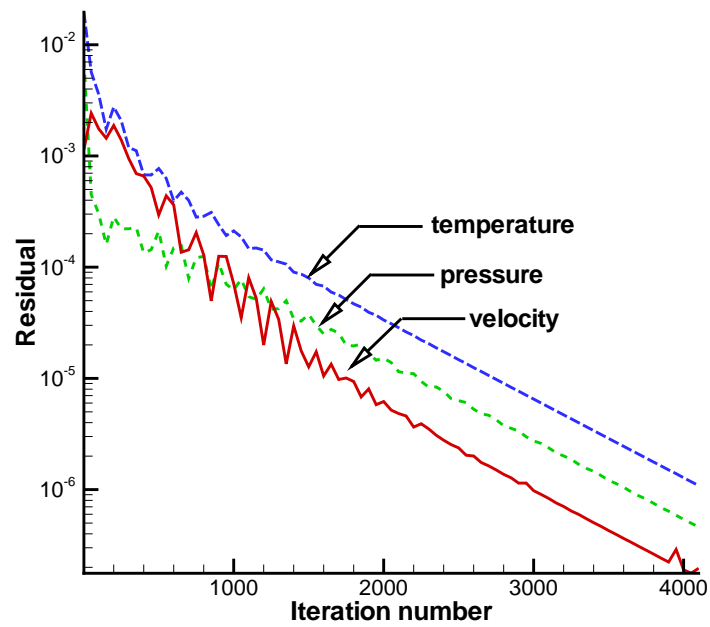


Figure 4.12: Schematic of the natural convection problem considered in Sec. 4.7.3.

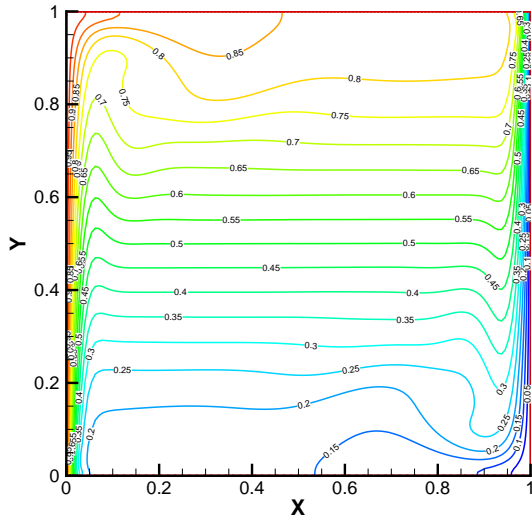


(a)

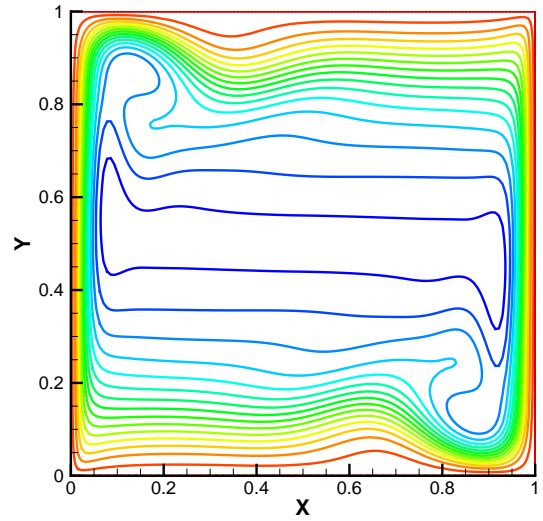


(b)

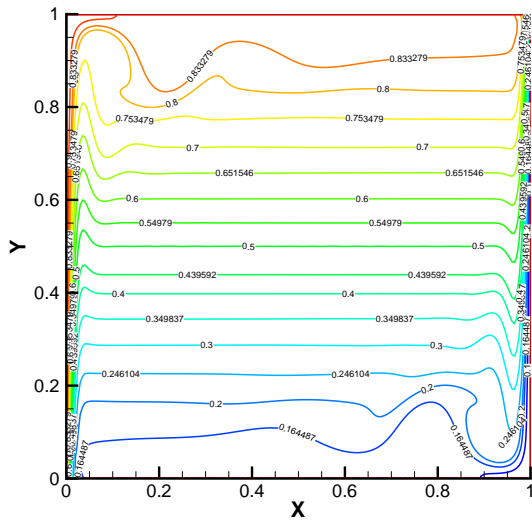
Figure 4.13: Residual reduction plots for the natural convection problem considered in Sec. 4.7.3 (a) $Ra = 10^7$; (b) $Ra = 10^8$



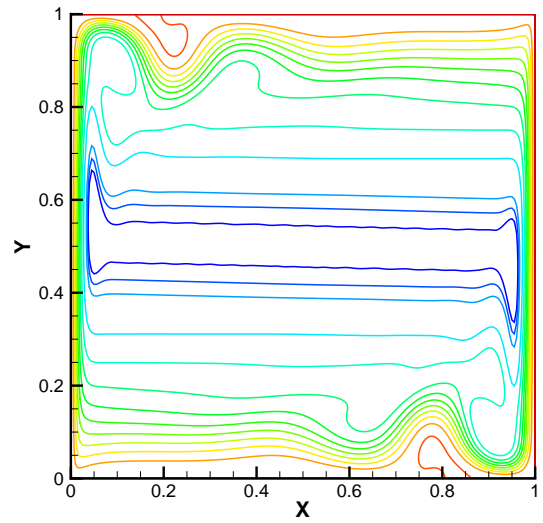
(a)



(b)

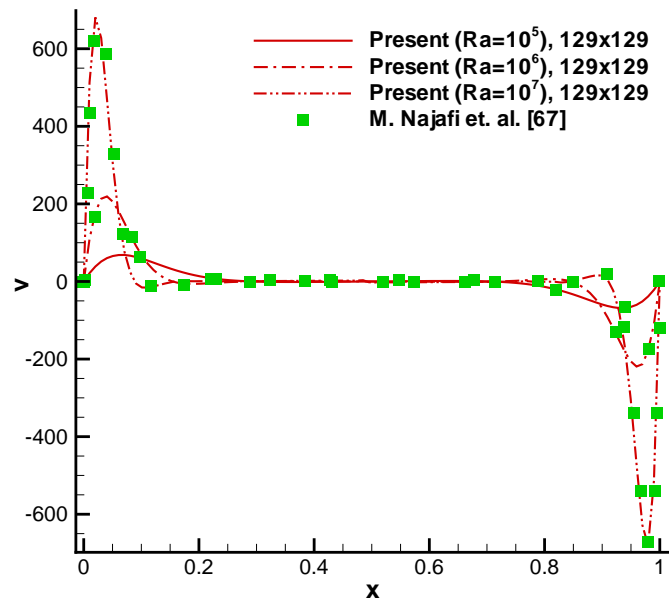


(c)

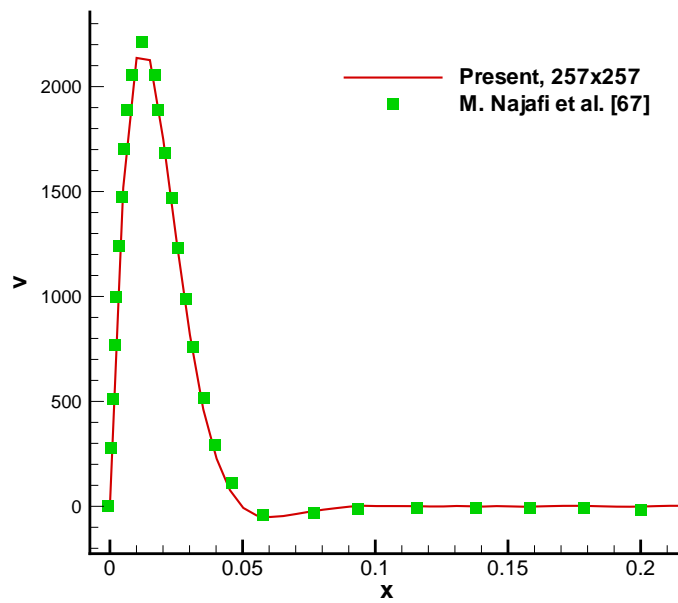


(d)

Figure 4.14: The streamfunction and T -temperature contours for the natural convection problem considered in Sec. 4.7.3. (a)-(b) $Ra = 10^7$; (c)-(d) $Ra = 10^8$.



(a)



(b)

Figure 4.15: Comparisons of the v -velocity profiles with the results of Najafi [67] for the natural convection problem considered in Sec. 4.7.3. (a) $Ra = 10^5 \sim 10^7$; (b) $Ra = 10^8$.

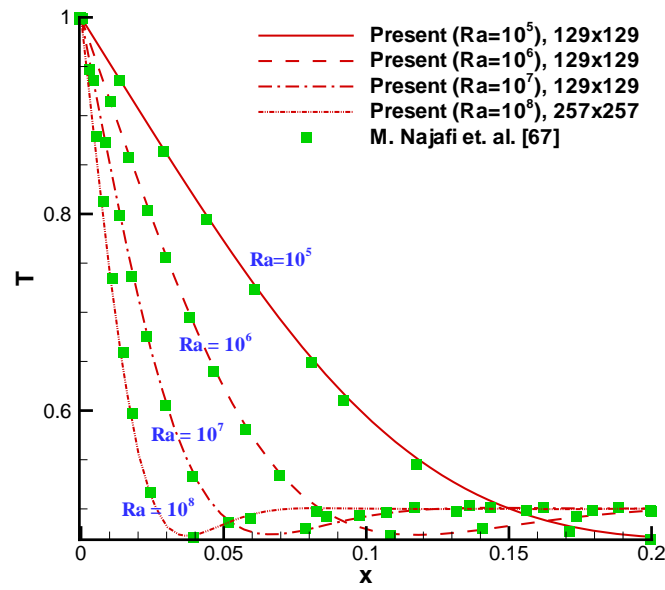


Figure 4.16: Comparison of the T -temperature profiles for $10^5 \leq Ra \leq 10^8$ with the results of Najafi [67] for the natural convection problem considered in Sec. 4.7.3.

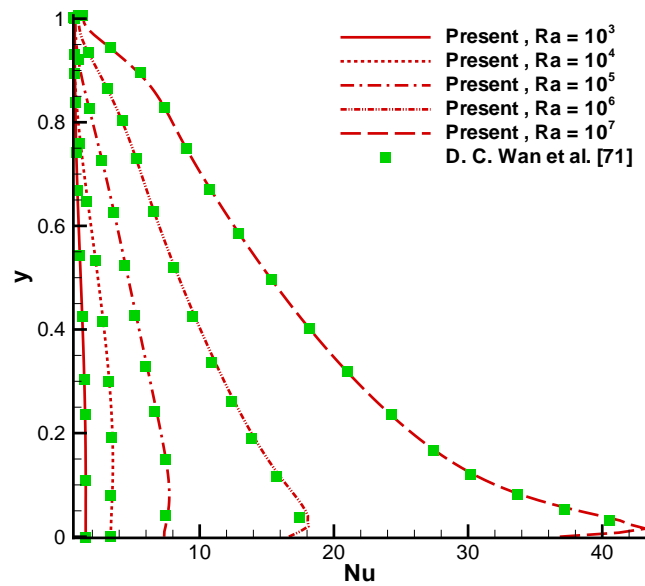


Figure 4.17: Comparison of the local Nusselt number distribution at the hot wall for $10^3 \leq Ra \leq 10^7$ with the results of Wan [71] for the natural convection problem considered in Sec. 4.7.3.



Chapter 5

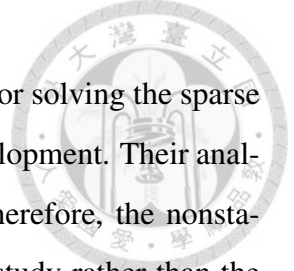
Iterative solver for three-dimensional Navier-Stokes finite element equations

5.1 Introduction

It is well known that matrix equation arising from the mixed finite element formulation for steady-state incompressible viscous Navier-Stokes equations lacks symmetry and definiteness. As a result, matrix equation becomes less diagonally dominant because of the presence of many diagonal zeros in the global stiffness matrix. These diagonal zeros owing to the continuity equation can further increase the matrix condition number. A complex distribution of eigenvalues for the matrix equation can furthermore cause the matrix equation to become indefinite. These difficulties pose a grand computational challenge.

The Gaussian-elimination-based direct solver has long been considered as the only way to circumvent the above difficulties. Unfortunately, the demand on the continued fill-in in the calculation of three-dimensional finite element solution from a direct solver is infeasible even if one uses state-of-the-art storage technology. Therefore, application of direct solver may no longer be practical. For this reason, there is a strong need to use iterative solver. As noted earlier, there is a trend toward using a robust iterative solver as research moves toward three-dimensional flow problems. The iterative solver is a strong rival to its direct counterpart because it is less prone to fill-in problems.

The iterative solver can be grouped into stationary and nonstationary type methods [72]. Stationary methods (e.g. Jacobi, Gaussian-Seidel, SOR) are older, simpler to



understand and implement. However, they are usually not effective for solving the sparse matrix equations. Nonstationary methods are a relatively recent development. Their analysis are harder to understand, but they can be highly effective. Therefore, the nonstationary or called Krylov subspace methods will be adopted in this study rather than the stationary methods. The Krylov subspace methods search for the solution of matrix equations $\underline{\underline{\mathbf{A}}}\underline{\underline{\mathbf{x}}} = \underline{\underline{\mathbf{b}}}$ in a certain subspace of the whole domain called the Krylov subspace. The Krylov subspace is spanned by vectors of polynomial of $\underline{\underline{\mathbf{A}}}$ of the form $\mathcal{P}(\underline{\underline{\mathbf{A}}})\underline{\underline{\mathbf{r}}}$, where \mathcal{P} and $\underline{\underline{\mathbf{r}}} = \underline{\underline{\mathbf{b}}} - \underline{\underline{\mathbf{A}}}\underline{\underline{\mathbf{x}}}$ represent the polynomial function and the residual, respectively. The iterative solver then project the solution onto a Krylov subspace of increased dimensions to converge towards the solution in each iteration.

It is known that the convergence behavior of iterative solvers depends greatly on the condition number of coefficient matrix. If the condition number is large, it will make the convergence behavior become very slow or even divergent. Hence, iterative method usually involves a transformation matrix to transform the coefficient matrix into the one with a smaller condition number. This transformation matrix is called the pre-conditioner. A good preconditioner can improve convergence of the iterative solver, sufficiently to overcome the extra cost of constructing and applying the pre-conditioner. Indeed, the iterative solver may fail without a pre-conditioner.

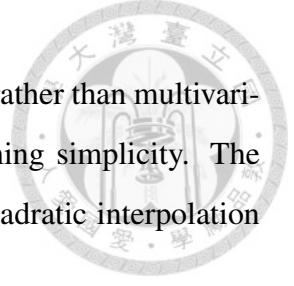
5.2 Interpolation functions

Let $\Omega \in \mathbb{R}^3$ be an open and bounded problem domain and $\partial\Omega = \Gamma$ denotes its boundary. The steady incompressible Navier-Stokes equations cast in primitive-variable form is given by

$$\nabla \cdot \underline{\underline{\mathbf{u}}} = 0 \tag{5.1}$$

$$(\underline{\underline{\mathbf{u}}} \cdot \nabla)\underline{\underline{\mathbf{u}}} + \nabla p - \frac{1}{Re} \nabla^2 \underline{\underline{\mathbf{u}}} = \underline{\underline{\mathbf{f}}} \tag{5.2}$$

where $\underline{\underline{\mathbf{u}}} = \{\underline{\underline{\mathbf{u}}}, \underline{\underline{\mathbf{v}}}, \underline{\underline{\mathbf{w}}}\}$ denotes the velocity vector, p the pressure, Re the Reynolds number and $\underline{\underline{\mathbf{f}}}$ the source term.



In three-dimensional finite element calculations, univariant [73] rather than multivariant element plotted in Fig. 5.1 is adopted because of its programming simplicity. The primitive velocity vector is approximated using the following tri-quadratic interpolation functions N_j ($j = 1 \sim 27$)


$$N_j = \left(\frac{3}{2} \bar{\xi}^2 + \frac{1}{2} \bar{\xi} + 1 + \xi^2 - \xi_j^2 \right) \left(\frac{3}{2} \bar{\eta}^2 + \frac{1}{2} \bar{\eta} + 1 + \eta^2 - \eta_j^2 \right) \left(\frac{3}{2} \bar{\zeta}^2 + \frac{1}{2} \bar{\zeta} + 1 + \zeta^2 - \zeta_j^2 \right). \quad (5.3)$$

where $\bar{\xi} = \xi\xi_j$, $\bar{\eta} = \eta\eta_j$ and $\bar{\zeta} = \zeta\zeta_j$ represent the normalized coordinates of the j -th node. In the mixed finite element context, the pressure unknown is approximated using the following tri-linear interpolation functions M_l ($l = 1 \sim 8$) in order to satisfy the \mathcal{LBB} condition [45–47].

$$M_l = \frac{1}{8} (1 + \bar{\xi})(1 + \bar{\eta})(1 + \bar{\zeta}) \quad (5.4)$$

5.3 The finite element formulation

The finite element matrix equation is obtained by substituting the finite element approximation $\underline{\mathbf{u}}^h = \sum_{j=1}^{27} u_j N_j$ and $\underline{\mathbf{p}}^h = \sum_{l=1}^8 p_l M_l$ for the respective $\underline{\mathbf{u}}$ and $\underline{\mathbf{p}}$ into the weighted residuals statement of Eqs. (5.1)-(5.2). A sparse and unsymmetric indefinite matrix system $\underline{\underline{\mathbf{A}}} \underline{\underline{\mathbf{x}}} = \underline{\underline{\mathbf{b}}}$ can be derived as



$$\underline{\underline{\mathbf{A}}} = \int_{\Omega} \begin{pmatrix} C_{ij} & 0 & 0 & -M_l \frac{\partial N_i}{\partial x_1} \\ 0 & C_{ij} & 0 & -M_l \frac{\partial N_i}{\partial x_2} \\ 0 & 0 & C_{ij} & -M_l \frac{\partial N_i}{\partial x_3} \\ M_l \frac{\partial N_j}{\partial x_1} & M_l \frac{\partial N_j}{\partial x_2} & M_l \frac{\partial N_j}{\partial x_3} & 0 \end{pmatrix} d\Omega, \quad (5.5)$$

$$\underline{\mathbf{x}} = \{\mathbf{u}_j, \mathbf{v}_j, \mathbf{w}_j, \mathbf{p}_l\} \quad (5.6)$$

$$\underline{\mathbf{b}} = - \int_{\Gamma_{out}} N_i \begin{pmatrix} p\mathbf{n}_x - \frac{1}{Re} \frac{\partial \mathbf{u}_j}{\partial \mathbf{n}} \\ p\mathbf{n}_y - \frac{1}{Re} \frac{\partial \mathbf{v}_j}{\partial \mathbf{n}} \\ p\mathbf{n}_z - \frac{1}{Re} \frac{\partial \mathbf{w}_j}{\partial \mathbf{n}} \\ 0 \end{pmatrix} d\Gamma. \quad (5.7)$$

where $\underline{\mathbf{n}} = \{\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z\}$ represents the unit outer normal vector. The component C_{ij} in Eq. (5.5) contains the biased part $B_i = \tau(N_m \tilde{u}_m) \frac{\partial N_i}{\partial x_k}$ which can be expressed as follows

$$C_{ij} = (N_i + B_i)(N_m \tilde{u}_m) \frac{\partial N_j}{\partial x_k} + \frac{1}{Re} \frac{\partial N_i}{\partial x_k} \frac{\partial N_j}{\partial x_k}. \quad (5.8)$$

The stabilization parameter τ in B_i is determined via Eq. (3.18) in order to minimize the wavenumber error of the convection term. In each element, the unsymmetric matrix equations shown in Eq. (5.5) needs to be calculated sequentially. All these elementary matrices are than mapped into their appropriate global row and column index to form a large-sized global matrix equations.

5.4 Newton linearization of the nonlinear convection term

To calculate the elementary matrix Eq. (5.5), the nonlinear convection term needs to be linearized first. Since the linearization of convection term will significantly affect the rate of convergence toward the final solutions. The choice of an appropriate linearization method is, thus, an important topic in computational fluid dynamics. In three-dimensional calculations, the Newton linearization [74] is employed to accelerate the nonlinear iteration of the incompressible Navier-Stokes equations



Taking the x-component momentum equation in Eq. (5.2) as example.

$$uu_x + vu_y + wu_z = -p_x + \frac{1}{Re}(u_{xx} + u_{yy} + u_{zz}) + f_x \quad (5.9)$$

Linearization of the convection term on the left-hand side of Eq. (5.9) starts from rewriting it as

$$(u^2)_x + (vu)_y + (wu)_z = -p_x + \frac{1}{Re}(u_{xx} + u_{yy} + u_{zz}) + f_x \quad (5.10)$$

Consider a function st , we can expand it in Taylor series about the current value and terminate the expansion after the first-derivative term to obtain

$$\begin{aligned} s^{k+1}t^{k+1} &= s^k t^k + \left[\frac{\partial}{\partial s}(st) \right]^k (s^{k+1} - s^k) + \left[\frac{\partial}{\partial t}(st) \right]^k (t^{k+1} - t^k) + \text{H.O.T.} \\ &= s^{k+1}t^k + s^k t^{k+1} - s^k t^k + \text{H.O.T.} \end{aligned} \quad (5.11)$$

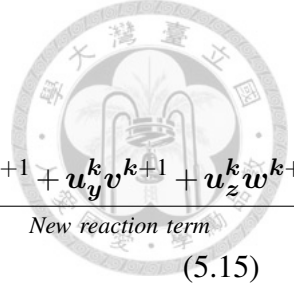
In the following derivation, all variables denoted by superscript k are evaluated using solution obtained from the previous nonlinear iteration. For those denoted by superscript $k+1$, they are evaluated at the most updated nonlinear iteration and are, therefore, referred to as the activity quantities. According to Eq. (5.11), the quantities $(u^2)_x$, $(vu)_y$ and $(wu)_z$ can be expressed as follows :

$$\begin{aligned} (u^2)_{x^{k+1}} &= (u^{k+1}u^k + u^k u^{k+1} - u^k u^k)_x \\ &= u_x^{k+1}u^k + u^{k+1}u_x^k + \underline{u_x^k u^{k+1}} + u^k u_x^{k+1} - \underline{u_x^k u^k} - u^k u_x^k \end{aligned} \quad (5.12)$$

$$\begin{aligned} (vu)_{y^{k+1}} &= (v^{k+1}u^k + v^k u^{k+1} - v^k u^k)_y \\ &= (v_y^{k+1}u^k + v^{k+1}u_y^k + \underline{v_y^k u^{k+1}} + v^k u_y^{k+1} - \underline{v_y^k u^k} - v^k u_y^k) \end{aligned} \quad (5.13)$$

$$\begin{aligned} (wu)_{z^{k+1}} &= (w^{k+1}u^k + v^k u^{k+1} - v^k u^k)_z \\ &= w_z^{k+1}u^k + w^{k+1}u_z^k + \underline{w_z^k u^{k+1}} + w^k u_z^{k+1} - \underline{w_z^k u^k} - w^k u_z^k \end{aligned} \quad (5.14)$$

Substituting Eqs. (5.12)-(5.14) into Eq. (5.10) and applying the continuity equation, one



can obtain the linearized x-momentum equation

$$\begin{aligned}
 & \underline{u}^k \underline{u}_x^{k+1} + \underline{v}^k \underline{u}_y^{k+1} + \underline{w}^k \underline{u}_z^{k+1} - \frac{1}{Re} (\underline{u}_{xx}^{k+1} + \underline{u}_{yy}^{k+1} + \underline{u}_{zz}^{k+1}) + \underline{u}_x^k \underline{u}^{k+1} + \underline{u}_y^k \underline{v}^{k+1} + \underline{u}_z^k \underline{w}^{k+1} \\
 & \quad \underline{+ p}_x^{k+1} = \underline{f}_x + \underline{u}^k \underline{u}_x^k + \underline{v}^k \underline{u}_y^k + \underline{w}^k \underline{u}_z^k \\
 & \hspace{10em} \text{New reaction term} \\
 & \hspace{10em} \text{New source term}
 \end{aligned} \tag{5.15}$$

The y- and z-component linearized momentum equations are obtained by following the similar derivation. Neglect of the underlined terms from the Newton linearization equations results in the convective lagging coefficient linearized equations.

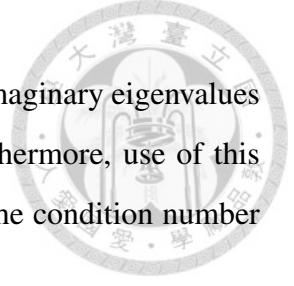
Prior to applying the iterative solver to solve a large-sized matrix equation, it is instructive to perform calculations in the smallest size of 2^3 tri-quadratic elements. The global matrix structure in the size of 2^3 elements is shown in Fig. 5.2. The red diamond, green circle, blue square and black triangle symbols are contributed from the momentum equations, continuity equation, Dirichlet boundary condition and Newton linearization procedure, respectively. It is clearly seen the presence of diagonal zeros due to the use of mixed finite element formulation.

5.5 The normalization procedure

It is known that no Krylov subspace iterative solver can be used to obtain unconditionally convergent solution from the unsymmetric and indefinite finite element matrix equations for the convection dominated flow problems [75]. In particular, this is true if the system involves a complex eigenvalue distribution and has a large condition number. For these reasons, the original unsymmetric indefinite finite element matrix equations $\underline{\underline{\mathbf{A}}} \underline{\underline{\mathbf{x}}} = \underline{\underline{\mathbf{b}}}$ can be transformed into its equivalent SPD counterpart by multiplying the transpose matrix $\underline{\underline{\mathbf{A}}}^T$ on both sides to obtain

$$\underline{\underline{\tilde{\mathbf{A}}}} \underline{\underline{\mathbf{x}}} = \underline{\underline{\tilde{\mathbf{b}}}} \tag{5.16}$$

where $\underline{\underline{\tilde{\mathbf{A}}}} = \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}}$ and $\underline{\underline{\tilde{\mathbf{b}}}} = \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{b}}}$. This procedure is called the normalization procedure [72] and the resulting SPD system is called the normal matrix equations. However, the normalization procedure significantly alters the distribution of eigenvalues. More precisely, the



nature of the matrix is changed from an indefinite matrix featuring imaginary eigenvalues to a positive definite matrix containing only real eigenvalues. Furthermore, use of this procedure largely increases the condition number in the sense that the condition number of $\underline{\tilde{\mathbf{A}}}$ is the square of the condition number of the original matrix $\underline{\mathbf{A}}$.

Since the convergence behavior of iterative solver strictly depends on the condition number. The increase of the condition number due to the normalization procedure makes the convergence of iterative solver become very slow or even divergent. To overcome this difficulty, the matrix equations need to be pre-conditioned using a properly chosen pre-conditioner. However, the traditional pre-conditioner is assumed to be in a global form (e.g. ILU). The lack of matrix assembling procedure will lead to the pre-conditioning difficulty. To overcome this problem, Hughes proposed the element-by-element pre-conditioner [72]. Unfortunately, we deal with the normalized matrix equations instead of the original ones, the value of elementary matrix $(\underline{\mathbf{A}}^T \underline{\mathbf{A}})_e$ is not available. In this study, two polynomial-based pre-conditioners are employed to reduce the condition number of normalized matrix equations.

5.6 The element-by-element technique

The application of the finite element method for the solution of Eqs. (5.1)-(5.2) usually results in a sparse and ill-conditioned global coefficient matrix. Storing the full global coefficient matrix will occupy a large amount of RAM memory. In order to reduce the memory demands, one recommended way is to store the non-zero entries of the global matrix in some sparse storage formats. Some sparse matrix formats, such as CSR (compressed sparse row), HYB (Hybrid) and JAD (Jagged Diagonal) [72], store the non-zero entries into an one-dimensional array and use some auxiliary arrays to indicate the locations of the non-zero entries. However, the use of sparse matrix format still requires a large amount of RAM memory in three-dimensional finite element calculation because the number of non-zero entries increases significantly as the problem size increases.

To overcome this difficulty, Hughes et al. proposed the memory-efficient element-by-

element (EBE) technique to avoid assembling the global matrix [76, 77]. The underlying idea of the EBE technique is based on the recognition that the assembling of elementary matrices to form the global matrix is a linear operation. The global coefficient matrix is obtained by sum up the all elementary matrices via the Boolean connectivity matrix. Wang et al. implemented the EBE-BICGSTAB to solve the three-dimensional incompressible Navier-Stokes equations [78]. Tezduyar combined the EBE and GMRES solver to solve the fluid-structure interaction problem [79]. Phoon developed the generalized Jacobi pre-conditioner to implement the CG solver together with the EBE strategy [80].

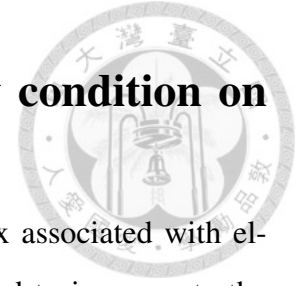
In the EBE context, the elementary matrices associated with each element are stored and never assembled. Contrary to the sparse global matrix $\underline{\underline{\mathbf{A}}}$, the elementary matrix $\underline{\underline{\mathbf{A}}}_e$ of size $n_e \times n_e$ (n_e being the local degree of freedom) is usually dense.

Applying the EBE concept, the most time-consuming global matrix-vector product operation $\underline{\underline{\mathbf{A}}}\mathbf{v}$ for a given vector \mathbf{v} in any Krylov subspace solver can be converted to calculations in terms of element-wise calculations. We denote by $\underline{\underline{\mathbf{B}}}_e$ the Boolean connectivity matrix, which maps the entries of e-th elementary matrix $\underline{\underline{\mathbf{A}}}_e$ into a global matrix $\underline{\underline{\mathbf{A}}}$. The global matrix-vector product $\underline{\underline{\mathbf{A}}}\mathbf{v}$ can be reformulated as follows

$$\underline{\underline{\mathbf{A}}}\mathbf{v} = \sum_{e=1}^{Nel} (\underline{\underline{\mathbf{B}}}_e)^T \underline{\underline{\mathbf{A}}}_e \underline{\underline{\mathbf{B}}}_e \mathbf{v} = \sum_{e=1}^{Nel} (\underline{\underline{\mathbf{B}}}_e)^T (\underline{\underline{\mathbf{A}}}_e \mathbf{v}_e)$$

where Nel is the number of total elements. A global matrix-vector product can be, as a result, represented by the operation carried out at element level for $\underline{\underline{\mathbf{A}}}_e \mathbf{x}_e$. This means that the product of global matrix and a vector is equivalent to the assembled vector of the elementary matrix-vector product. Any Krylov subspace iterative method applied in conjunction with the EBE technique has, therefore, a much lower memory demands. Because of this advantage, a large-scaled three-dimensional finite element flow calculation can be carried out on a relative modest desktop. One more advantage of the EBE technique worth mentioning is that no global numbering skill (e.g. RCM [81]) is required in order to minimize the global matrix bandwidth.

5.7 Implementation of Dirichlet boundary condition on elementary matrix



Since the global matrix is never assembled, the elementary matrix associated with elements containing the boundary nodes must be, therefore, modified to incorporate the Dirichlet boundary conditions. Note that the resulting elementary matrices must yield the same effect of the global matrix obtained from the assembly procedure and Dirichlet boundary condition implementation.

We denote by $\underline{\underline{\mathbf{A}}}_e$ the e -th elementary matrix and the superscript b means that this element contains the Dirichlet boundary condition node. For the sake of simplicity, we take a domain containing the shaded Dirichlet boundary region shown in Fig. 5.3 as an example. In Fig. 5.3, the number shown in the circle and the square symbol represent the *global* and *local* numbering, respectively. Each global corner nodes, namely, 1, 7, 19 and 25 shares only one element, which is element 1, 2, 3 and 4, respectively. The middle edge nodes (4,10,16,22) share with two elements, respectively. The node 13 is shared by four elements. Thus for each boundary node (1,4,7,10,13,16,19,22,25), the data of (i) the ID of boundary element; (ii) the number of shared boundary node and (iii) the corresponding local row index of the boundary element matrix $\underline{\underline{\mathbf{A}}}_e^b$, are required. Using these data, the boundary element matrix $\underline{\underline{\mathbf{A}}}_e^b$ and global right hand side vector can be, therefore, modified for the inclusion of the Dirichlet boundary values.

The above procedure can be explained by considering the nodes 4, which are shared by elements 1 and 2. The boundary elementary matrix $\underline{\underline{\mathbf{A}}}_1^b$ is given by

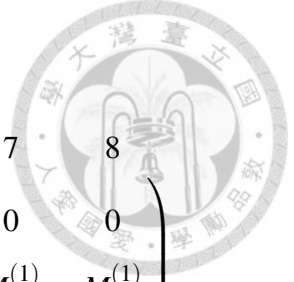


$$\underline{\underline{\mathbf{A}}}_1^b = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \begin{pmatrix} M_{11}^{(1)} & M_{12}^{(1)} & M_{13}^{(1)} & M_{14}^{(1)} & M_{15}^{(1)} & M_{16}^{(1)} & M_{17}^{(1)} & M_{18}^{(1)} \\ M_{21}^{(1)} & M_{22}^{(1)} & M_{23}^{(1)} & M_{24}^{(1)} & M_{25}^{(1)} & M_{26}^{(1)} & M_{27}^{(1)} & M_{28}^{(1)} \\ M_{31}^{(1)} & M_{32}^{(1)} & M_{33}^{(1)} & M_{34}^{(1)} & M_{35}^{(1)} & M_{36}^{(1)} & M_{37}^{(1)} & M_{38}^{(1)} \\ M_{41}^{(1)} & M_{42}^{(1)} & M_{43}^{(1)} & M_{44}^{(1)} & M_{45}^{(1)} & M_{46}^{(1)} & M_{47}^{(1)} & M_{48}^{(1)} \\ M_{51}^{(1)} & M_{52}^{(1)} & M_{53}^{(1)} & M_{54}^{(1)} & M_{55}^{(1)} & M_{56}^{(1)} & M_{57}^{(1)} & M_{58}^{(1)} \\ M_{61}^{(1)} & M_{62}^{(1)} & M_{63}^{(1)} & M_{64}^{(1)} & M_{65}^{(1)} & M_{66}^{(1)} & M_{67}^{(1)} & M_{68}^{(1)} \\ M_{71}^{(1)} & M_{72}^{(1)} & M_{73}^{(1)} & M_{74}^{(1)} & M_{75}^{(1)} & M_{76}^{(1)} & M_{77}^{(1)} & M_{78}^{(1)} \\ M_{81}^{(1)} & M_{82}^{(1)} & M_{83}^{(1)} & M_{84}^{(1)} & M_{85}^{(1)} & M_{86}^{(1)} & M_{87}^{(1)} & M_{88}^{(1)} \end{pmatrix} \end{matrix}$$

In Fig. 5.3, the boundary node 4 appears in both the elements 1 and 2 but with different row index. It appears in the third row in the element 1 whereas it appears in the first row in the element 2. As discussed by Reddy [11], the global matrix and right hand side vector must be modified for the known value at the Dirichlet boundary node. This can be achieved by enforcing the following conditions in $\underline{\underline{\mathbf{A}}}_1^b$

$$\begin{cases} M_{11}^{(1)} = 1.0/S_1^{(1)} \\ M_{1j}^{(1)} = 0, j = 2 \sim 8 \\ \underline{\underline{\mathbf{b}}}(1) = \phi_1 \end{cases}$$

where $S_1^{(1)} = 1$ in this example denotes the number of shared boundary node 1 of element 1 and ϕ_1 is the Dirichlet boundary value of boundary node 1. Repeating this modification procedure for other Dirichlet boundary nodes in the element 1, the resulting modified boundary elementary matrix $\underline{\underline{\mathbf{A}}}_1^b$ is given by

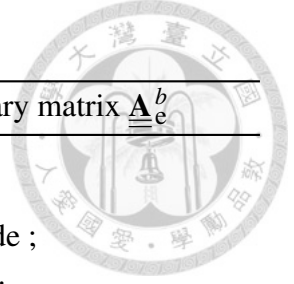


$$\underline{\underline{\mathbf{A}}}_1^b = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \left(\begin{array}{cccccccc} 1/S_1^{(1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M_{21}^{(1)} & M_{22}^{(1)} & M_{23}^{(1)} & M_{24}^{(1)} & M_{25}^{(1)} & M_{26}^{(1)} & M_{27}^{(1)} & M_{28}^{(1)} \\ 0 & 0 & 1/S_3^{(1)} & 0 & 0 & 0 & 0 & 0 \\ M_{41}^{(1)} & M_{42}^{(1)} & M_{43}^{(1)} & M_{44}^{(1)} & M_{45}^{(1)} & M_{46}^{(1)} & M_{47}^{(1)} & M_{48}^{(1)} \\ 0 & 0 & 0 & 0 & 1/S_5^{(1)} & 0 & 0 & 0 \\ M_{61}^{(1)} & M_{62}^{(1)} & M_{63}^{(1)} & M_{64}^{(1)} & M_{65}^{(1)} & M_{66}^{(1)} & M_{67}^{(1)} & M_{68}^{(1)} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/S_7^{(1)} & 0 \\ M_{81}^{(1)} & M_{82}^{(1)} & M_{83}^{(1)} & M_{84}^{(1)} & M_{85}^{(1)} & M_{86}^{(1)} & M_{87}^{(1)} & M_{88}^{(1)} \end{array} \right) \end{matrix}$$

where $S_3^{(1)} = S_5^{(1)} = 2$ and $S_7^{(1)} = 4$. For the element 2, the modified elementary matrix $\underline{\underline{\mathbf{A}}}_2^b$ is also given by

$$\underline{\underline{\mathbf{A}}}_2^b = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \left(\begin{array}{cccccccc} 1/S_1^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M_{21}^{(2)} & M_{22}^{(2)} & M_{23}^{(2)} & M_{24}^{(2)} & M_{25}^{(2)} & M_{26}^{(2)} & M_{27}^{(2)} & M_{28}^{(2)} \\ 0 & 0 & 1/S_3^{(2)} & 0 & 0 & 0 & 0 & 0 \\ M_{41}^{(2)} & M_{42}^{(2)} & M_{43}^{(2)} & M_{44}^{(2)} & M_{45}^{(2)} & M_{46}^{(2)} & M_{47}^{(2)} & M_{48}^{(2)} \\ 0 & 0 & 0 & 0 & 1/S_5^{(2)} & 0 & 0 & 0 \\ M_{61}^{(2)} & M_{62}^{(2)} & M_{63}^{(2)} & M_{64}^{(2)} & M_{65}^{(2)} & M_{66}^{(2)} & M_{67}^{(2)} & M_{68}^{(2)} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/S_7^{(2)} & 0 \\ M_{81}^{(2)} & M_{82}^{(2)} & M_{83}^{(2)} & M_{84}^{(2)} & M_{85}^{(2)} & M_{86}^{(2)} & M_{87}^{(2)} & M_{88}^{(2)} \end{array} \right) \end{matrix}$$

where $S_3^{(2)} = 1$, $S_1^{(2)} = S_7^{(2)} = 2$ and $S_5^{(2)} = 4$. In general, if ϕ_{bn} is the Dirichlet boundary value at the boundary node 'bn', the modification procedure can be stated in the Algorithm 5.1. Note that the computational task of this algorithm is insignificant because the number of boundary nodes is relatively small compared to the total number of grid points in the problem domain



Algorithm 5.1: The modification procedure on boundary elementary matrix $\underline{\underline{\mathbf{A}}}_e^b$

```

1  $Nel^b$  : The number of boundary elements ;
2  $n_e$  : The degree of freedom of elementary matrix ;
3  $e$  : The ID of element containing the Dirichlet boundary node ;
4  $S_k^{(e)}$  : The number of shared boundary node  $k$  in the element  $e$  ;
5  $\phi_{bn}$  : The Dirichlet boundary value at the boundary node 'bn' ;
6  $M_{ij}^{(e)}$  : The  $(i, j)$ -component of matrix  $\underline{\underline{\mathbf{A}}}_e^b$  ;
7  $\mathbf{b}$  : The global right hand side vector ;
8 for  $e = 1 \rightarrow Nel^b$  do
9   Find the local row index  $r$  and global row index  $i$  for each boundary node  $k$  in
   the element  $e$  ;
10  Set  $M_{rr}^{(e)} = 1.0/S_k^{(e)}$  ;
11  for  $j = 1 \rightarrow n_e, j \neq r$  do
12     $M_{rj}^{(e)} = 0$ 
13  end
14  Set  $\mathbf{b}(i) = \phi_{bn}$  ;
15 end

```

After modifying the all boundary elementary matrices $\underline{\underline{\mathbf{A}}}_e^b$ via Algorithm (5.1), it is easy to verify that the assembled modified elementary matrices will produce the same effect of the global matrix. This procedure can be easily applied to modify the elementary matrices discretized from the incompressible Navier-Stokes equations Eqs. (5.1)-(5.2) using the mixed finite element formulation.

5.8 Implementation of polynomial-based pre-conditioner

To reduce the condition number without deterioration of prediction accuracy, the Eq. (5.16) should be pre-conditioned prior to the calculation of solution through iterative solver. In this study, two pre-conditioners belonging to the class of polynomial pre-conditioners for normalized matrix equations will be derived.

Start with the SPD normal matrix equations, we can rewrite it as $\underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{D}}} \left[\underline{\underline{\mathbf{I}}} - \left(\underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{D}}}^{-1} \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}} \right) \right]$ or $\left(\underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}} \right)^{-1} = \left[\underline{\underline{\mathbf{I}}} - \left(\underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{D}}}^{-1} \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}} \right) \right]^{-1} \underline{\underline{\mathbf{D}}}^{-1}$, where $\underline{\underline{\mathbf{D}}}$ denotes the diagonal part of $\underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}}$. Define $\underline{\underline{\mathbf{G}}} = \underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{D}}}^{-1} \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}}$, the matrix $\underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}}$ can be expressed in terms of the $\underline{\underline{\mathbf{G}}}$ as $\left(\underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}} \right)^{-1} = \left(\underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{G}}} \right)^{-1} \underline{\underline{\mathbf{D}}}^{-1}$. Given this matrix $\underline{\underline{\mathbf{G}}}$, one can rewrite $\left(\underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{G}}} \right)^{-1}$

analytically in a form of binomial series as $(\underline{\mathbf{I}} - \underline{\mathbf{G}})^{-1} = \sum_{k=0}^{\infty} \underline{\mathbf{G}}^k$ [72], provided that the spectral radius of $\underline{\mathbf{G}}$ is less than one. Hence, the inverse of the SPD matrix $\underline{\mathbf{A}}^T \underline{\mathbf{A}}$ can be approximated analytically in terms of the following infinite series of matrix summation

$$\left(\underline{\mathbf{A}}^T \underline{\mathbf{A}}\right)^{-1} = \sum_{k=0}^{\infty} \underline{\mathbf{G}}^k \underline{\mathbf{D}}^{-1} \quad (5.17)$$

Approximation of $(\underline{\mathbf{A}}^T \underline{\mathbf{A}})^{-1}$ by choosing $k = 0$ and $k = 1$ leads respectively to $\underline{\mathbf{D}}^{-1}$ and $\underline{\mathbf{G}} \underline{\mathbf{D}}^{-1}$, which correspond to pre-conditioning the matrix $\underline{\mathbf{A}}^T \underline{\mathbf{A}}$ by employing the well-known Jacobi pre-conditioner $(\underline{\mathbf{M}}_J)^{-1} = \underline{\mathbf{D}}^{-1}$ and the polynomial pre-conditioner $(\underline{\mathbf{M}}_P)^{-1} = 2\underline{\mathbf{D}}^{-1} - \underline{\mathbf{D}}^{-1} \underline{\mathbf{A}}^T \underline{\mathbf{A}} \underline{\mathbf{D}}^{-1}$, respectively.

Note that polynomial pre-conditioner can be employed, provided that the spectral radius of the matrix $\underline{\mathbf{G}}$ is less than one. For this reason, the distribution of eigenvalues of $\underline{\mathbf{G}}$ in the size of 2^3 elements is plotted in Fig. 5.4, which shows that not all the eigenvalues of $\underline{\mathbf{G}}$ lie in the unit circle. This means that the spectral radius is greater than one, implying that Eq. (5.17) may not hold. To make the spectral radius of $\underline{\mathbf{G}}$ smaller than one, the so-called scaling parameter ω [72] is introduced for the approximation of $\underline{\mathbf{A}}^T \underline{\mathbf{A}}$. In the derivation, we start with $\omega \underline{\mathbf{A}}^T \underline{\mathbf{A}}$ instead of $\underline{\mathbf{A}}^T \underline{\mathbf{A}}$. Repeating the above derivation procedure, the inverse of $\underline{\mathbf{A}}^T \underline{\mathbf{A}}$ can be derived as $\omega(\underline{\mathbf{I}} - \widehat{\underline{\mathbf{G}}}) \underline{\mathbf{D}}^{-1}$, where $\widehat{\underline{\mathbf{G}}} = \underline{\mathbf{I}} - \omega \underline{\mathbf{D}}^{-1} \underline{\mathbf{A}}^T \underline{\mathbf{A}}$. By applying the binomial series, the scaling polynomial pre-conditioner is derived as

$$\left(\underline{\mathbf{M}}_{SP}\right)^{-1} = \omega \left(\underline{\mathbf{I}} - \widehat{\underline{\mathbf{G}}}\right) \underline{\mathbf{D}}^{-1} = 2\omega \underline{\mathbf{D}}^{-1} - \omega^2 \underline{\mathbf{D}}^{-1} \underline{\mathbf{A}}^T \underline{\mathbf{A}} \underline{\mathbf{D}}^{-1} \quad (5.18)$$

The remaining work is to determine the user's specific scaling parameter ω in a way that the spectral radius of $\widehat{\underline{\mathbf{G}}}$ is less than one. For choosing a proper range of ω , as before the finite element calculation is conducted in the size of 2^3 elements. In Fig. 5.5, the spectral radius ρ is plotted with respect to the scaling parameter ω for the matrix equation $\widehat{\underline{\mathbf{G}}}$. It can be clearly seen that the spectral radius is less than one in the range of $0 \leq \omega \leq 0.1$. In this study, $\omega = 0.05$ is chosen in the rest of our all calculations. For completeness, the eigenvalues calculated at $\omega = 0.05$ are also plotted in Fig. 5.6. It is clearly seen the spectral radius of $\widehat{\underline{\mathbf{G}}}$ is less than one.



5.9 Iterative matrix solver

Due to the unsymmetric and ill-conditioned properties of Eq. (5.5), it is a common practice to solve the matrix equations using the direct solver. However, the use of direct solver for three-dimensional problem produces prohibitively high computational costs. According to author's experience, the direct solver is suitable for two-dimensional problem whereas iterative solver is more practical for three-dimensional problem. The reason is that storage demands will be prohibitive in fill-in processed using direct solver with the increasing of nodal points size.

Before investigating the different Krylov subspace iterative method, a basic idea of the iterative solver is introduced firstly. Let $\underline{\underline{\mathbf{D}}}$ be the nonsingular diagonal part of $\underline{\underline{\mathbf{A}}}$ and $\underline{\underline{\mathbf{I}}}$ be the identity matrix, one can apply the matrix-splitting operator to rewrite the system $\underline{\underline{\mathbf{A}}}\mathbf{x} = \underline{\underline{\mathbf{b}}}$ as

$$\mathbf{x} = \underline{\underline{\mathbf{B}}}\mathbf{x} + \underline{\underline{\mathbf{b}}}$$
 where $\underline{\underline{\mathbf{B}}} = \underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{D}}}^{-1}\underline{\underline{\mathbf{A}}}$, $\underline{\underline{\mathbf{b}}} = \underline{\underline{\mathbf{D}}}^{-1}\underline{\underline{\mathbf{b}}}$

and we obtain the following iteration formulation.

$$\mathbf{x}_{n+1} = \underline{\underline{\mathbf{B}}}\mathbf{x}_n + \underline{\underline{\mathbf{b}}} \quad (5.19)$$

It is well known that the Eq. (5.19) will converge to exact solution $\hat{\mathbf{x}} = \underline{\underline{\mathbf{A}}}^{-1}\underline{\underline{\mathbf{b}}}$ if and only if the spectral radius $\rho(\underline{\underline{\mathbf{B}}})$ is less than one [82]. Unfortunately, even the simplest boundary value problem, like the Laplace equation, discretized by Galerkin finite element model, shows that the Eq. (5.19) may converge very slowly.

Since the exact solution $\hat{\mathbf{x}}$ is unknown, we can not compute the error at the n -th iteration step

$$\underline{\underline{\mathbf{d}}}_n = \mathbf{x}_n - \hat{\mathbf{x}}$$

In order to check the convergence, the n -th residual vector is normally employed.

$$\mathbf{r}_n = \underline{\underline{\mathbf{b}}} - \underline{\underline{\mathbf{A}}}\mathbf{x}_n = -\underline{\underline{\mathbf{A}}}\underline{\underline{\mathbf{d}}}_n \quad (5.20)$$



Assuming $\underline{\mathbf{D}} = \underline{\mathbf{I}}$ and letting $\underline{\mathbf{B}} = \underline{\mathbf{I}} - \underline{\mathbf{A}}$, we have

$$\underline{\mathbf{r}}_n = \underline{\mathbf{b}} - \underline{\mathbf{A}}\underline{\mathbf{x}}_n = \underline{\mathbf{B}}\underline{\mathbf{x}}_n + \underline{\mathbf{b}} - \underline{\mathbf{x}}_n = \widehat{\underline{\mathbf{B}}}\underline{\mathbf{x}}_n + \widehat{\underline{\mathbf{b}}} - \underline{\mathbf{x}}_n = \underline{\mathbf{x}}_{n+1} - \underline{\mathbf{x}}_n \quad (5.21)$$

Hence, the iteration Eq. (5.19) can be rewritten as

$$\underline{\mathbf{x}}_{n+1} = \underline{\mathbf{x}}_n + \underline{\mathbf{r}}_n \quad (5.22)$$

Multiplying the Eq. (5.22) by $-\underline{\mathbf{A}}$ yields a recursive formulation for the residual.

$$\underline{\mathbf{r}}_{n+1} = \underline{\mathbf{r}}_n - \underline{\mathbf{A}}\underline{\mathbf{r}}_n = \underline{\mathbf{B}}\underline{\mathbf{r}}_n \quad (5.23)$$

One can compute $\underline{\mathbf{r}}_n$ either according to the definition Eq. (5.21) or by Eq. (5.23). From Eq. (5.23), one can get by induction

$$\underline{\mathbf{r}}_n = \mathcal{P}_n(\underline{\mathbf{A}})\underline{\mathbf{r}}_0 \in \text{span}\{\underline{\mathbf{r}}_0, \underline{\mathbf{A}}\underline{\mathbf{r}}_0, \underline{\mathbf{A}}^2\underline{\mathbf{r}}_0, \dots, \underline{\mathbf{A}}^n\underline{\mathbf{r}}_0\} \quad (5.24)$$

where $\mathcal{P}_n(\zeta) = (1 - \zeta)^n$ is a polynomial of degree n . From Eq. (5.22), we have

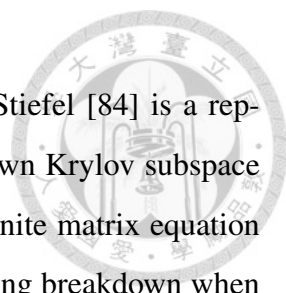
$$\underline{\mathbf{x}}_n = \underline{\mathbf{x}}_0 + \underline{\mathbf{r}}_0 + \dots + \underline{\mathbf{r}}_{n-1} = \underline{\mathbf{x}}_0 + \mathcal{P}_{n-1}(\underline{\mathbf{A}})\underline{\mathbf{r}}_0 \quad (5.25)$$

with a polynomial \mathcal{P}_{n-1} of degree $n - 1$. The Eq. (5.25) shows that $\underline{\mathbf{x}}_n$ lies in the affine space $\underline{\mathbf{x}}_0 + \text{span}\{\underline{\mathbf{r}}_0, \dots, \underline{\mathbf{A}}^{n-1}\underline{\mathbf{r}}_0\}$ and $\mathcal{K}_n = \mathcal{K}_n(\underline{\mathbf{A}}, \underline{\mathbf{r}}_0) := \text{span}\{\underline{\mathbf{x}}_0, \dots, \underline{\mathbf{A}}^{n-1}\underline{\mathbf{r}}_0\}$ is called the Krylov subspace [83]. The calculation of $\mathcal{P}_{n-1}(\underline{\mathbf{A}})\underline{\mathbf{r}}_0$ requires $n - 1$ matrix-vector product operations because one needs to build up the subspace $\text{span}\{\underline{\mathbf{r}}_0, \dots, \underline{\mathbf{A}}^{n-1}\underline{\mathbf{r}}_0\}$. This can be done by multiplying the vector $\underline{\mathbf{r}}_0$ by the matrix $\underline{\mathbf{A}}$ and using the right hand side to get $\underline{\mathbf{A}}^2$ and so on.

We denote by \mathcal{K} and \mathcal{L} the search space and constraint subspace, respectively. The idea behind Krylov subspace solver is to approximate the solution $\underline{\mathbf{x}}$ from $\underline{\mathbf{A}}\underline{\mathbf{x}} = \underline{\mathbf{b}}$ iteratively via a sequence of $\{\underline{\mathbf{x}}_1, \underline{\mathbf{x}}_2, \dots\}$ so that the corresponding residuals $\underline{\mathbf{r}}_n (= \underline{\mathbf{b}} - \underline{\mathbf{A}}\underline{\mathbf{x}}_n)$ is orthogonal to constraint subspace. i.e.

$$\text{Find } \underline{\mathbf{x}}_n \in \underline{\mathbf{x}}_0 + \mathcal{K} \text{ s.t. } \underline{\mathbf{b}} - \underline{\mathbf{A}}\underline{\mathbf{x}}_n \perp \mathcal{L}$$

The Krylov subspace methods can be classified into the so-called orthogonal method ($\mathcal{L} = \mathcal{K}$) and non-orthogonal method ($\mathcal{L} \neq \mathcal{K}$). Krylov subspace method belong to both



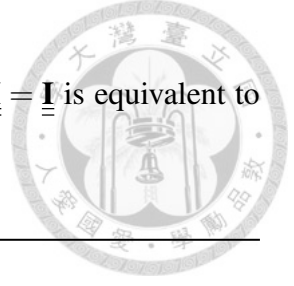
methods. The Conjugate Gradient (CG) method of Hestenes and Stiefel [84] is a representation of orthogonal methods. It is the oldest and the best known Krylov subspace method. It works effectively only for a symmetric and positive definite matrix equation containing a clustered set of real eigenvalues and suffers from pivoting breakdown when matrix symmetry is lost.

The BiConjugate Gradient (BICG) solver [85] was designed to handle the matrix asymmetry. As with the CG solver, the BICG solver terminates in at most n -steps for $n \times n$ matrix equation. The disadvantage is that there is no minimization property in BICG solver and it results in the irregular convergence behavior. Other inherent deficiency is the need of performing the matrix transpose operation. The Biconjugate Gradient Stabilized method (BICGSTAB) of Van Der Vorst [86] improves the convergence behavior of the BICG and no transpose matrix operation is required. The General Minimum RESidual (GMRES) proposed by Saad [87] is the best well known non-orthogonal Krylov subspace method. It is a generalization of a method called MINRES (MINimal RESidual) method which can be used to solve nonsymmetric matrix equations

Since the CG solver has long been recognized as the optimal solver due to its ability of obtaining the unconditionally convergent solution from symmetric and positive definite (SPD) matrix equations. One possible cure for convergence difficulty when solving the unsymmetric matrix equations Eqs. (5.5)-(5.7) is to deal with its equivalent SPD matrix equations via normalization procedure. The CG solver can be therefore applied to get the unconditionally convergent solution.

5.9.1 The pre-conditioned CG solver

The CG solver is the most well known of the Krylov subspace method for solving linear systems for the SPD matrix equations. As mentioned early, the constraint subspace for CG algorithm is chosen as $\mathcal{L}_n = \mathcal{K}_n$. The CG solver consists in generating a sequence of vectors converging towards the exact solution vector. The vectors are generated by searching a distance along a specific search direction, resulting in vectors being conjugate to each other. The EBE-based pre-conditioned CG solver for solving the normal equa-



tions in this study is stated in Algorithm 5.2. Note that choosing $\underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{I}}}$ is equivalent to the original CG solver.

Algorithm 5.2 : The PCG algorithm for $\underline{\underline{\mathbf{A}}} \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{x}}} = \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{b}}}$

Starting from an initial guess $\underline{\underline{\mathbf{x}}}_0$

Compute $\underline{\underline{\mathbf{x}}}'_0 = \sum_e \underline{\underline{\mathbf{A}}}_e \underline{\underline{\mathbf{x}}}_0$, $\underline{\underline{\mathbf{x}}}''_0 = \sum_e \underline{\underline{\mathbf{A}}}_e^T \underline{\underline{\mathbf{x}}}'_0$, $\underline{\underline{\mathbf{b}}}' = \sum_e \underline{\underline{\mathbf{A}}}_e^T \underline{\underline{\mathbf{b}}}$ \leftarrow **EBE procedure**

$\underline{\underline{\mathbf{p}}}_0 = \underline{\underline{\mathbf{z}}}_0$

For $j = 1, 2, \dots$, $\underline{\underline{\mathbf{p}}}'_{j-1} = \sum_e (\underline{\underline{\mathbf{A}}}_e \underline{\underline{\mathbf{p}}}_{j-1})$ \leftarrow **EBE procedure**

$\underline{\underline{\mathbf{p}}}''_{j-1} = \sum_e (\underline{\underline{\mathbf{A}}}_e^T \underline{\underline{\mathbf{p}}}'_{j-1})$ \leftarrow **EBE procedure**

$\alpha_{j-1} = (\underline{\underline{\mathbf{p}}}_{j-1}, \underline{\underline{\mathbf{r}}}_{j-1}) / (\underline{\underline{\mathbf{p}}}_{j-1}, \underline{\underline{\mathbf{p}}}''_{j-1})$

$\underline{\underline{\mathbf{x}}}_j = \underline{\underline{\mathbf{x}}}_{j-1} + \alpha_{j-1} \underline{\underline{\mathbf{p}}}_{j-1}$

$\underline{\underline{\mathbf{r}}}_j = \underline{\underline{\mathbf{r}}}_{j-1} - \alpha_{j-1} \underline{\underline{\mathbf{p}}}''_{j-1}$

Convergence check

$\underline{\underline{\mathbf{z}}}_j = \underline{\underline{\mathbf{M}}}^{-1} \underline{\underline{\mathbf{r}}}_j$

\leftarrow $\underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{M}}}_J$ or $\underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{M}}}_{SP}$

$\beta_{j-1} = (\underline{\underline{\mathbf{z}}}_j, \underline{\underline{\mathbf{r}}}_j) / (\underline{\underline{\mathbf{r}}}_{j-1}, \underline{\underline{\mathbf{r}}}_{j-1})$

$\underline{\underline{\mathbf{p}}}_j = \underline{\underline{\mathbf{z}}}_j + \beta_{j-1} \underline{\underline{\mathbf{p}}}_{j-1}$

end

In Algorithm 5.2, $\underline{\underline{\mathbf{p}}}$ is the search direction vector and α is the search distance along the vector $\underline{\underline{\mathbf{p}}}$ for each iteration. The $\underline{\underline{\mathbf{p}}}_{j+1}$ and residual $\underline{\underline{\mathbf{r}}}_{j+1}$ of the Algorithm 5.2 are enforced to be orthogonal to all previous $\underline{\underline{\mathbf{p}}}_j$ and $\underline{\underline{\mathbf{r}}}_j$. Moreover, the coefficient α and β are used to minimize $(\underline{\underline{\mathbf{x}}}_j - \hat{\underline{\underline{\mathbf{x}}}}, \underline{\underline{\mathbf{A}}}(\underline{\underline{\mathbf{x}}}_j - \hat{\underline{\underline{\mathbf{x}}}}))$ in the affine space $\underline{\underline{\mathbf{x}}}_0 + \mathcal{K}_n$ where $\hat{\underline{\underline{\mathbf{x}}}}$ is the exact solution. In general, this minimum is guaranteed to exist if the coefficient matrix is SPD. The CG algorithm is attractive due to its cheap computational cost per iteration and low memory requirement.



5.9.2 The pre-conditioned BICGSTAB solver

Since the objective in this study is to advocate the use of PCG solver for solving the normal system arising from the incompressible Navier-Stokes equations to get the unconditionally convergent solution. For the sake of comparison, the highly recommended BICGSTAB solver is also applied to solve the original unsymmetric and indefinite matrix equations. The BICGSTAB is a variant of CG which, in contrast to CG, can be successfully applied to solve the general matrix equations. The CG can only keep orthogonal residual vectors for the SPD matrix equations. The idea behind the BICGSTAB is substituting the orthogonal sequence of residual by two mutually orthogonal sequence. It is a projection process onto the search subspace

$$\mathcal{K}_n = \text{span}\{\underline{\mathbf{v}}, \underline{\mathbf{A}}\underline{\mathbf{v}}, \underline{\mathbf{A}}^2\underline{\mathbf{v}}, \dots, \underline{\mathbf{A}}^{n-1}\underline{\mathbf{v}}\}$$

and orthogonal to the constraint subspace.

$$\mathcal{L}_n = \text{span}\{\underline{\boldsymbol{\gamma}}, \underline{\mathbf{A}}^T\underline{\boldsymbol{\gamma}}, (\underline{\mathbf{A}}^T)^2\underline{\boldsymbol{\gamma}}, \dots, (\underline{\mathbf{A}}^T)^{n-1}\underline{\boldsymbol{\gamma}}\}$$

where $\underline{\boldsymbol{\gamma}} = \underline{\mathbf{r}}_0 / \|\underline{\mathbf{r}}_0\|$ and $\underline{\mathbf{r}}_0$ denotes the initial residual vector. The vector $\underline{\mathbf{v}}$ in \mathcal{K}_n can be chosen arbitrarily such that the inner product $(\underline{\mathbf{v}}, \underline{\boldsymbol{\gamma}}) \neq 0$. This solver is therefore considered for the assessment purpose. The algorithm of the EBE-based pre-conditioned BICGSTAB (PBICGSTAB) is stated in Algorithm 5.3.

Since no global minimum is guaranteed, the Algorithm 5.3 may lead to irregular convergence behavior [78] and it occasionally fail to find a solution. The breakdown seems to occur rarely in practice. The advantage of Algorithm 5.3 is that it still posses the attractive features of CG method.

Algorithm 5.3 : The PBICGSTAB algorithm for $\underline{\mathbf{A}} \underline{\mathbf{x}} = \underline{\mathbf{b}}$

Starting from an initial guess $\underline{\mathbf{x}}_0$

Compute the initial residual $\underline{\mathbf{r}}_0 = \underline{\mathbf{b}} - \underline{\mathbf{A}}\underline{\mathbf{x}}_0$

$\underline{\mathbf{M}}$ represents the pre-conditioner



Choose $\underline{\mathbf{v}}$, such that $(\underline{\mathbf{v}}', \underline{\mathbf{r}}_0) \neq 0$

For $j = 1, 2, \dots$,

$$\rho_{j-1} = (\underline{\mathbf{v}}, \underline{\mathbf{r}}_{j-1})$$

if $\rho_{j-1} < \varepsilon_1$ (**near breakdown**)

if $j = 1$

$$\underline{\mathbf{p}}_j = \underline{\mathbf{r}}_{j-1}$$

else

$$\beta_{j-1} = (\rho_{j-1} / \rho_{j-2}) / (\alpha_{j-1} / \omega_{j-1})$$

$$\underline{\mathbf{p}}_j = \underline{\mathbf{r}}_{j-1} + \beta_{j-1} (\underline{\mathbf{p}}_{j-1} - \omega_{j-1} \underline{\mathbf{v}}_{j-1})$$

end if

Solve $\underline{\mathbf{M}} \underline{\mathbf{p}}'_j = \underline{\mathbf{p}}_j$

$\underline{\mathbf{v}}_j = \sum_c (\underline{\mathbf{A}}_c \underline{\mathbf{p}}'_j)$ \leftarrow **EBE procedure**

$$\alpha_j = \rho_{j-1} / (\underline{\mathbf{v}}, \underline{\mathbf{v}}_j)$$

if $(\underline{\mathbf{v}}, \underline{\mathbf{v}}_j) < \varepsilon_2$ (**near breakdown**)

$$\underline{\mathbf{s}}_j = \underline{\mathbf{r}}_{j-1} - \alpha_j \underline{\mathbf{v}}_j$$

Solve $\underline{\mathbf{M}} \underline{\mathbf{s}}'_j = \underline{\mathbf{s}}_j$

$\underline{\mathbf{t}} = \sum_c (\underline{\mathbf{A}}_c \underline{\mathbf{s}}'_j)$ \leftarrow **EBE procedure**

if $\|\underline{\mathbf{s}}_j\| < \varepsilon$

$$\omega_j = 0$$

else

$$\omega_j = (\underline{\mathbf{t}}, \underline{\mathbf{s}}) / (\underline{\mathbf{t}}, \underline{\mathbf{t}})$$

end if

$\underline{\mathbf{x}}_j = \underline{\mathbf{x}}_{j-1} + \alpha_j \underline{\mathbf{p}}'_j + \omega_j \underline{\mathbf{s}}'_j$ \leftarrow **Update**

$$\underline{\mathbf{r}}_j = \underline{\mathbf{s}}_j - \omega_j \underline{\mathbf{t}}$$

Convergence check

end



5.9.3 The pre-conditioned GMRES solver

For the sake of completeness, the well-known GMRES solver, which is an extension of MINRES solver [72], is also adopted for solving the original matrix equations. The GMRES was first proposed by Sadd [87] and is based on the well-known Arnoldi procedure. It is an iterative method that approximates the solution of matrix equation by the vector in Krylov subspace with minimal residual. The Gram-Schmidt orthogonalization procedure is adopted in order to construct the orthogonal basis of the Krylov subspace. In GMRES, the search space and constraint subspace are taken respectively as $\mathcal{K} = \mathcal{K}_n$ and $\mathcal{L}_n = \underline{\underline{\mathbf{A}}}\mathcal{K}_n$. It can be proven that this choice of constraint subspace can minimize the norm of residual in affine subspace $\underline{\mathbf{x}}_0 + \mathcal{K}_n$. i.e.

$$\underline{\mathbf{x}}_n = \text{Min} \|\underline{\mathbf{b}} - \underline{\underline{\mathbf{A}}}\underline{\mathbf{x}}\| \forall \underline{\mathbf{x}} \in \underline{\mathbf{x}}_0 + \mathcal{K}_m$$

The main steps of pre-conditioned GMRES (PGMRES) implemented in an EBE fashion [88] are stated in Algorithm 5.4 :

Algorithm 5.4 : The PGMRES algorithm for $\underline{\underline{\mathbf{A}}}\underline{\mathbf{x}} = \underline{\mathbf{b}}$

$\underline{\underline{\mathbf{M}}}$ represents the pre-conditioner

Starting from an initial guess $\underline{\mathbf{x}}_0$

For $j = 1, 2, \dots,$

Solve $\underline{\underline{\mathbf{M}}}\underline{\mathbf{r}}_j = \underline{\mathbf{b}} - \underline{\underline{\mathbf{A}}}\underline{\mathbf{x}}_0$

$\underline{\mathbf{v}}_1 = \underline{\mathbf{r}}_j / \|\underline{\mathbf{r}}_j\|$

$\underline{\mathbf{s}} := \|\underline{\mathbf{r}}_j\| \underline{\mathbf{e}}_1$

For $i = 1, 2, \dots,$

$\underline{\mathbf{z}} = \underline{\underline{\mathbf{A}}}\underline{\mathbf{v}}_j \leftarrow$ **EBE procedure**

Solve $\underline{\underline{\mathbf{M}}}\underline{\mathbf{w}} = \underline{\mathbf{z}}$

For $k = 1, 2, \dots,$

$h_{k,i} = (\underline{\mathbf{w}}, \underline{\mathbf{v}}_k)$

$\underline{\mathbf{w}} = \underline{\mathbf{w}} - h_{k,i}\underline{\mathbf{v}}_k$



end

$$h_{i+1,i} = \|\underline{\mathbf{w}}\|$$

$$\underline{\mathbf{v}} = \underline{\mathbf{w}}/h_{i+1,i}$$

Apply J_1, \dots, J_{i-1} on $(h_{1,i}), \dots, (h_{i+1,i})$

Construct J_i , acting on the i -th and $(i+1)$ -th component of $h_{.,i}$

such that the $(i+1)$ -th component of $J_i h_{.,i}$ is zero

$$\underline{\mathbf{s}} := J_i \underline{\mathbf{s}}$$

if $\mathbf{s}(i+1)$ is small enough, then **UPDATE** $(\tilde{\underline{\mathbf{x}}}, i)$ and quit

end

UPDATE $(\tilde{\underline{\mathbf{x}}}, m)$

end

The **UPDATE** $(\tilde{\underline{\mathbf{x}}}, i)$ procedure is as follows

Solve $\underline{\mathbf{y}}$ from $\underline{\mathbf{H}} \underline{\mathbf{y}} = \tilde{\underline{\mathbf{s}}}$ in which the upper $i \times i$ triangular part of $\underline{\mathbf{H}}$ has $h_{i,j}$ as its elements.

$\tilde{\underline{\mathbf{s}}}$ represents the first i components of $\underline{\mathbf{s}}$

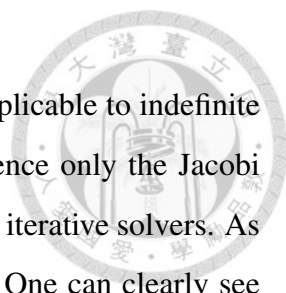
$$\tilde{\underline{\mathbf{x}}} = \underline{\mathbf{x}}_0 + \sum_{k=1}^i \underline{\mathbf{y}}_k \underline{\mathbf{v}}_k$$

Compute $\|\underline{\mathbf{b}} - \underline{\mathbf{A}} \tilde{\underline{\mathbf{x}}}\|_2$ **and check the convergence**

If $\tilde{\underline{\mathbf{x}}}$ is accurate enough, then terminate the calculation

else $\underline{\mathbf{x}}_0 = \tilde{\underline{\mathbf{x}}}$

In Algorithm 5.4, the vector $\underline{\mathbf{y}}_k$ is designed to minimize the residual $\|\underline{\mathbf{b}} - \underline{\mathbf{A}} \underline{\mathbf{x}}_n\|_2$. A drawback of GMRES is that one need to store the whole Krylov subspace, requiring a large amount of memory, and also increasing the work required per iteration. This can be circumvented by restarting the GMRES iteration. After the chosen restart number m , the accumulated data are cleared and the intermediate results are used as the initial condition for the next m iterations. This procedure is repeated until the convergence criterion is satisfied. This restart procedure decrease the memory requirement and deteriorates the convergence but works reasonably well in practice. The choice of m is a matter of experience. In this study, m is fixed as 5 for all computations.



It is worth noting that the polynomial pre-conditioner can not be applicable to indefinite matrix calculation using either the GMRES or the BICGSTAB. Hence only the Jacobi pre-conditioner can be implemented in the GMRES and BICGSTAB iterative solvers. As before, the finite element calculation in 2^3 elements is carried out. One can clearly see from Fig. 5.7 that the spectral radius can by no means be less than one.

5.10 Numerical results

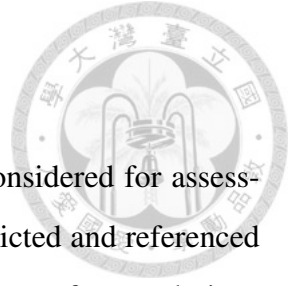
5.10.1 Verification study

The first step toward verification of the developed incompressible Navier-Stokes three-dimensional finite element code is to solve a problem amenable to analytical solution. The problem under investigation is defined in a hexahedron of length 1. Along the cube surfaces, the nodal velocities are analytically prescribed by $\mathbf{u} = \frac{1}{2}(\mathbf{y}^2 + \mathbf{z}^2)$, $\mathbf{v} = -z$, and $\mathbf{w} = \mathbf{y}$. The corresponding pressure solution takes the form as

$$\mathbf{p} = \frac{1}{2}(\mathbf{y}^2 + \mathbf{z}^2) + \frac{2}{Re} \mathbf{x}$$

In the finite element verification study, which involves N number of unknowns, the memory is estimated to be $O(N^{4/3})$ using the direct Frontal solver. Such a large memory demand prohibits us from getting an accurate solution effectively using the Frontal solver for large-scaled problem.

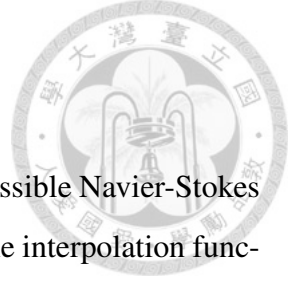
Three different element number are used to perform the convergence test. Table 5.1 tabulates the computed L_2 error norms, from which the iterative solutions are seen to be compatible with the solution by Frontal solver. The CPU times are summarized in Table 5.2. From these results, it is concluded that the direct Frontal solver is preferable to iterative solvers when solving a smaller-size problem. Much of the CPU time is consumed in the iterative solver. This points out the difficulty of solving the unsymmetric and indefinite matrix equations.



5.10.2 Lid-driven cavity flow in a cube

The lid-driven cavity problem schematically shown in Fig. 5.8 is considered for assessment of the proposed three iterative solvers. To begin with, the predicted and referenced mid-line velocity profiles [89] are plotted in Fig. 5.9. Good agreement of two solutions is shown. Since one of our objectives is to demonstrate the necessity of normalizing the matrix equations, the computed results using the BICGSTAB and GMRES solvers for the original matrix equations and the CG solver for the normal matrix equations at different nodal points are also tabulated in Tables 5.3-5.5. Unlike the other two solvers, steady solution can be stably obtained by the CG solver for each investigated case. The necessity of performing matrix normalization is therefore amply demonstrated.

While matrix normalization can be applied to get the steady-state solution using the PCG solver, it will also bring in two drawbacks. One is related to the increased condition number, and the other one is the necessity of performing additional matrix-vector multiplication. Since the convergence behavior of the Krylov subspace solver is very sensitive to the condition number, a large condition number makes the CG solver converge very slowly or even diverge. In Tables 5.3-5.5, the inner iteration number of the CG solver is greater than those of the BICGSTAB and GMRES solvers. To reduce the condition number and improve convergence, the pre-conditioning technique is adopted. The results of PCG used together with the Jacobi and polynomial pre-conditioners at different nodal points are shown in Tables 5.6-5.8. For the sake of completeness, the results obtained from the PBICGSTAB and PGMRES solver along with the use of Jacobi pre-conditioner are also shown in these tables. One can clearly see from Fig. 5.10 that for the case $Re = 400$, the number of inner iteration for the PCG solver together with the pre-conditioner is much smaller than the original CG, PBICGSTAB and PGMRES solver. As the Reynolds number is increased to 1000, one can see from Fig. 5.11 that the polynomial pre-conditioner outperforms the Jacobi pre-conditioner in reducing the inner iteration number within the context of the proposed PCG iterative solver.



5.11 Conclusion remarks

The current finite element calculation of three-dimensional incompressible Navier-Stokes equations has been carried out in tri-quadratic/tri-linear elements. The interpolation functions for the primitive variables \underline{u} and p satisfy the \mathcal{LBB} compatibility condition to circumvent the oscillatory pressure solution. To enhance numerical stability in association with the predicted velocity, the stabilized term is added along the streamline direction. To minimize the wavenumber error for the convection term, a proper upwinding coefficient is rigorously chosen. Prior to the calculation of solution from finite element matrix equations, which has been transformed to a SPD matrix through the normalization procedure of the original unsymmetric and indefinite mixed finite element equations. This procedure is essential to reduce the condition number of the normalized SPD matrix equations. Both of the Jacobi and polynomial pre-conditioners are investigated. To reduce the memory requirement, the element-by-element strategy is implemented in the iterative solver. The performance of three iterative solvers is also assessed. It is concluded that the use of CG iterative solver together with the pre-conditioner for solving the normal matrix equations is superior to the pre-conditioned GMRES and BICGSTAB iterative solvers for solving the original matrix equations.

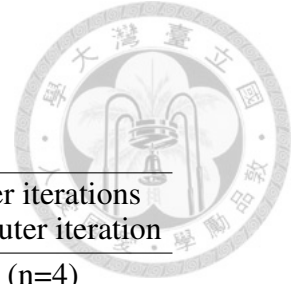


Solver		Number of elements		
		4^3	8^3	16^3
Frontal	u	1.532×10^{-3}	8.241×10^{-4}	—
	v	8.973×10^{-3}	5.489×10^{-3}	—
	w	9.338×10^{-3}	6.044×10^{-3}	—
	p	3.194×10^{-2}	2.069×10^{-2}	—
BICGSTAB	u	1.460×10^{-3}	8.253×10^{-4}	4.191×10^{-4}
	v	8.915×10^{-3}	5.535×10^{-3}	2.761×10^{-3}
	w	9.479×10^{-3}	6.004×10^{-3}	3.065×10^{-3}
	p	3.071×10^{-2}	1.477×10^{-2}	6.629×10^{-3}
GMRES	u	1.460×10^{-3}	8.253×10^{-4}	4.191×10^{-4}
	v	8.915×10^{-3}	5.535×10^{-3}	2.761×10^{-3}
	w	9.479×10^{-3}	6.004×10^{-3}	3.065×10^{-3}
	p	3.073×10^{-2}	1.481×10^{-3}	6.695×10^{-3}
CG	u	1.460×10^{-3}	8.254×10^{-4}	4.191×10^{-4}
	v	8.915×10^{-3}	5.535×10^{-3}	2.761×10^{-3}
	w	9.479×10^{-3}	6.004×10^{-3}	3.066×10^{-3}
	p	3.533×10^{-2}	1.481×10^{-2}	6.661×10^{-3}

Table 5.1: The computed L_2 error norms for the problem considered in Sec. 5.10.1. In this table, "—" means that the solution is not computable.

Solvers	Total CPU time (s)	CPU time and percentage in () for iterative solver
BICGSTAB	4,608.6	4,527.5 (98.24%)
GMRES	11,459.1	11,257.4 (98.23%)
CG	3,015.2	2,967.4 (98.71%)

Table 5.2: The total CPU times for three solvers in 16^3 tri-quadratic elements.

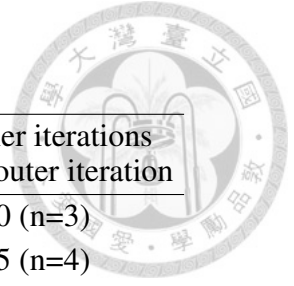


Solvers	Reynolds number	CPU time (s)	No. of outer iterations	No. of inner iterations at the n-th outer iteration
BICGSTAB	100	682.7	8	1,465 (n=4)
	400	×	×	Break down
	1000	×	×	Break down
GMRES	100	300.2	6	430 (n=3)
	400	1,238.4	8	1,350 (n=4)
	1000	2,623.3	9	2,120 (n=5)
CG	100	795.4	8	2,073 (n=4)
	400	862.2	8	2,788 (n=4)
	1000	1,937.7	10	5,463 (n=5)

Table 5.3: The finite element solutions computed from three iterative solvers for the problem considered in a domain of 21^3 nodal points. The notation "×" shown in this table means that the solution is not computable.

Solvers	Reynolds number	CPU time (s)	No. of outer iterations	No. of inner iterations at the n-th outer iteration
BICGSTAB	100	5,853.1	7	3,040 (n=3)
	400	30,622.5	8	5,585 (n=4)
	1000	×	×	Break down
GMRES	100	4,652.1	6	850 (n=3)
	400	12,468.4	7	1,760 (n=4)
	1000	×	×	Break down
CG	100	4,384.9	6	3,460 (n=3)
	400	9,490.4	7	4,428 (n=4)
	1000	29,845.3	11	7,611 (n=6)

Table 5.4: The finite element solutions computed from three iterative solvers for the problem considered in a domain of 41^3 nodal points. The notation "×" shown in this table means that the solution is not computable.

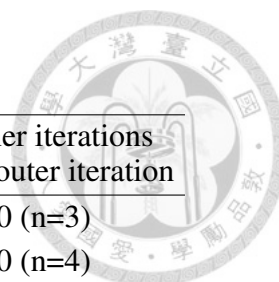


Solvers	Reynolds number	CPU time (s)	No. of outer iterations	No. of inner iterations at the n-th outer iteration
BICGSTAB	100	57,630.5	6	3,090 (n=3)
	400	172,855.2	8	9,995 (n=4)
	1000	×	×	Break down
GMRES	100	71,598.2	6	2,840 (n=3)
	400	151,312.7	7	4,070 (n=4)
	1000	×	×	Break down
CG	100	46,097.8	6	4,281 (n=3)
	400	88,192.9	8	6,001 (n=4)
	1000	272,705.7	12	10,657 (n=6)

Table 5.5: The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 61^3 nodal points. The notation "×" shown in this table means that the solution is not computable.

Solvers	Reynolds number	CPU time (s)	No. of outer iterations	No. of inner iterations at the n-th outer iteration
PBICGSTAB	100	294.6	6	1,400 (n=3)
	400	3,852.5	8	1,870 (n=4)
	1000	×	×	Break down
PGMRES	100	254.1	6	300 (n=3)
	400	1,033.5	8	930 (n=4)
	1000	2,094.9	9	1,470 (n=5)
PJCG	100	93.2	6	364 (n=3)
	400	362.5	8	1,230 (n=4)
	1000	1,466.9	15	2,631 (n=8)
PPCG	100	107.5	6	361 (n=3)
	400	425.4	8	1,229 (n=4)
	1000	1,051.4	9	2,600 (n=5)

Table 5.6: The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 21^3 nodal points. The notation "×" shown in this table means that the solution is not computable. PJCG : CG iterative solver used together with Jacobi pre-conditioner ; PPCG : CG iterative solver used together with polynomial pre-conditioner.



Solvers	Reynolds number	CPU time (s)	No. of outer iterations	No. of inner iterations at the n-th outer iteration
PBICGSTAB	100	5,510.3	6	2,600 (n=3)
	400	10,297.1	8	4,870 (n=4)
	1000	×	×	Break down
PGMRES	100	3,935.1	6	600 (n=3)
	400	10,537.0	7	1,610 (n=4)
	1000	×	×	Break down
PJCG	100	1,918.3	6	986 (n=3)
	400	4,248.3	7	1,960 (n=4)
	1000	13,167.7	9	5,017 (n=5)
PPCG	100	2,711.5	6	978 (n=3)
	400	5,799.5	7	1,498 (n=4)
	1000	12,230.8	9	3,301 (n=5)

Table 5.7: The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 41^3 nodal points. The notation "×" shown in this table means that the solution is not computable.

Solvers	Reynolds number	CPU time (s)	No. of outer iterations	No. of inner iterations at the n-th outer iteration
PBICGSTAB	100	49,805.3	7	5,455(n=4)
	400	98,022.3	8	6,615(n=4)
	1000	×	×	Break down
PGMRES	100	36,839.9	6	930(n=3)
	400	82,338.9	7	1,670(n=4)
	1000	×	×	Break down
PJCG	100	25,153.4	6	1,996 (n=3)
	400	46,770.4	10	2,149 (n=5)
	1000	172,560.3	15	6,218 (n=7)
PPCG	100	32,717.1	6	2,395 (n=3)
	400	51,790.1	7	2,960 (n=4)
	1000	133,386.8	11	5,329 (n=5)

Table 5.8: The finite element solutions computed from three iterative solvers for the problem investigated in a domain of 61^3 nodal points. The notation "×" shown in this table means that the solution is not computable.

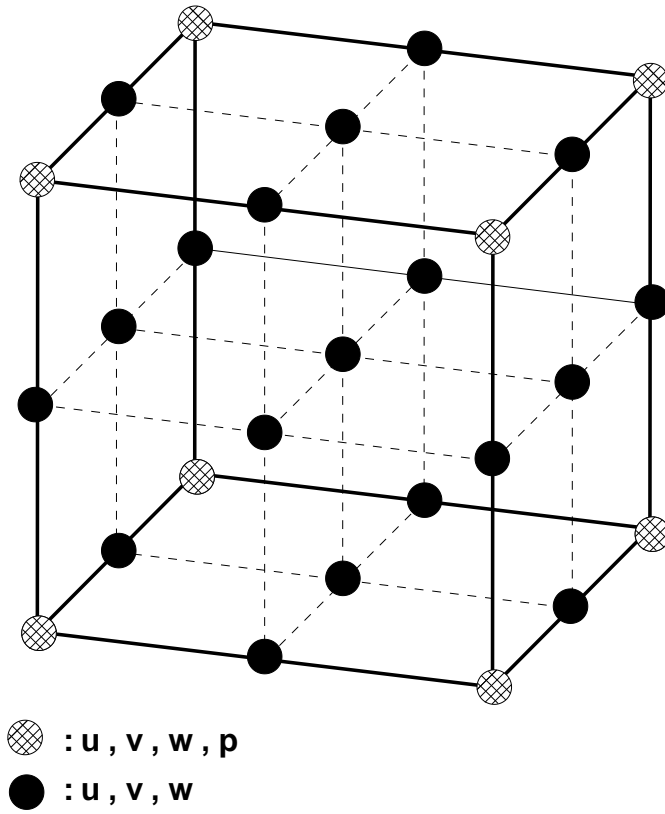
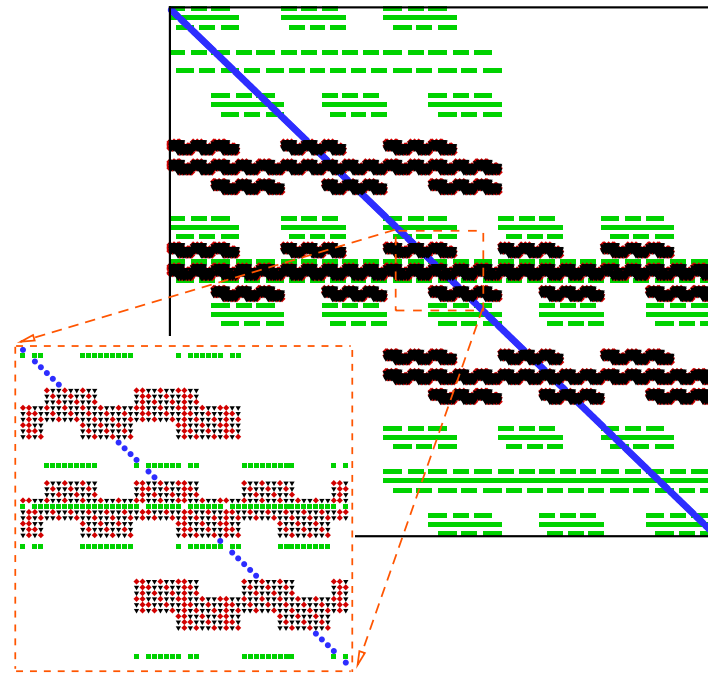
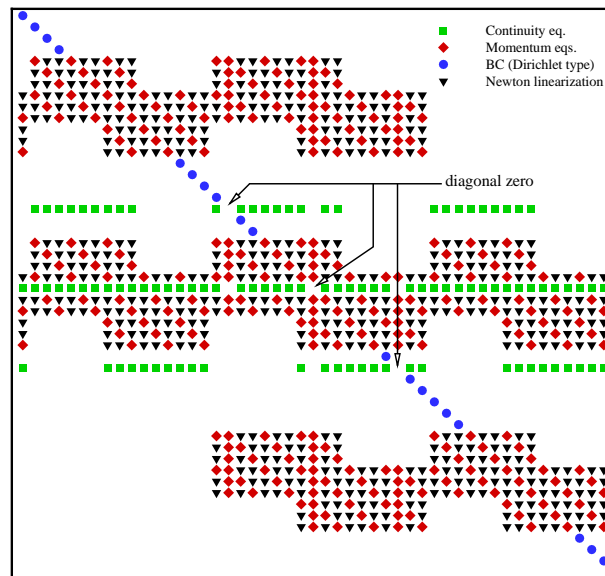


Figure 5.1: Schematic of the primitive variable storing in a tri-quadratic element



(a)



(b)

Figure 5.2: Illustration of a 402×402 finite element matrix equation derived in the 2^3 tri-quadratic elements. The green square, red diamond, blue circle and black triangle symbols represent the non-zero entries contributed from the continuity equation, momentum equations, Dirichlet-type boundary data and Newton linearization, respectively. (a) Global matrix profile; (b) Matrix profile in the dashed block of matrix (a).

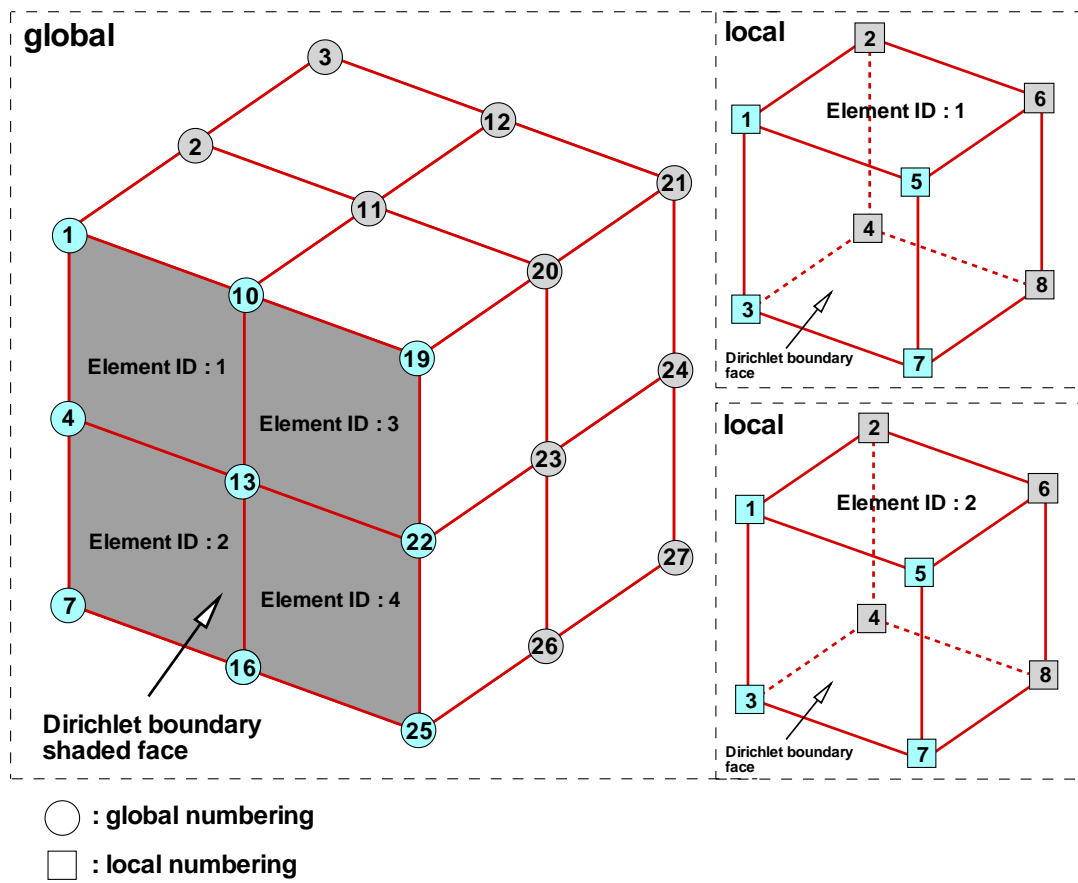


Figure 5.3: Illustration of the Dirichlet boundary condition implementation on boundary elementary matrix

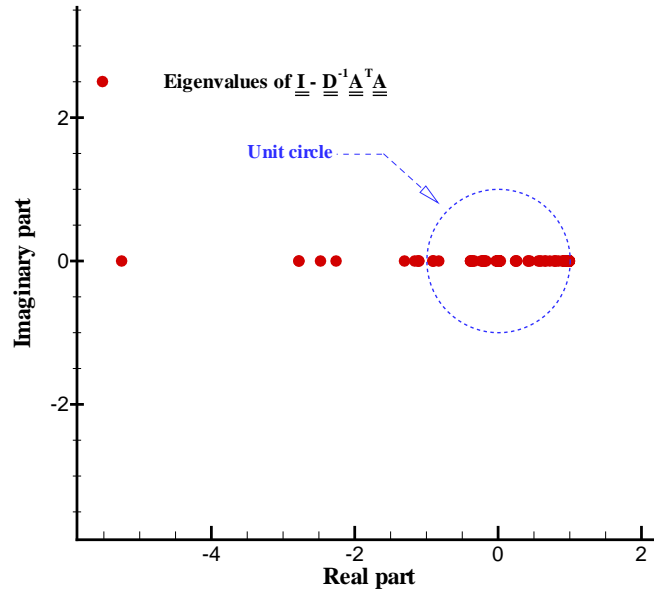


Figure 5.4: Computed eigenvalues for the matrix equation $\underline{\underline{I}} - \underline{\underline{D}}^{-1} \underline{\underline{A}}^T \underline{\underline{A}}$

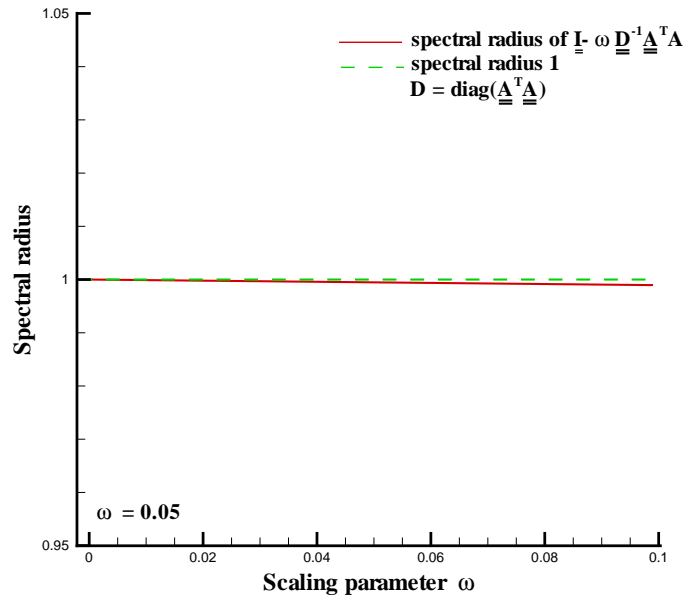


Figure 5.5: Predicted spectral radius of $\underline{\underline{I}} - \omega \underline{\underline{D}}^{-1} \underline{\underline{A}}^T \underline{\underline{A}}$ versus the scaling parameter ω .

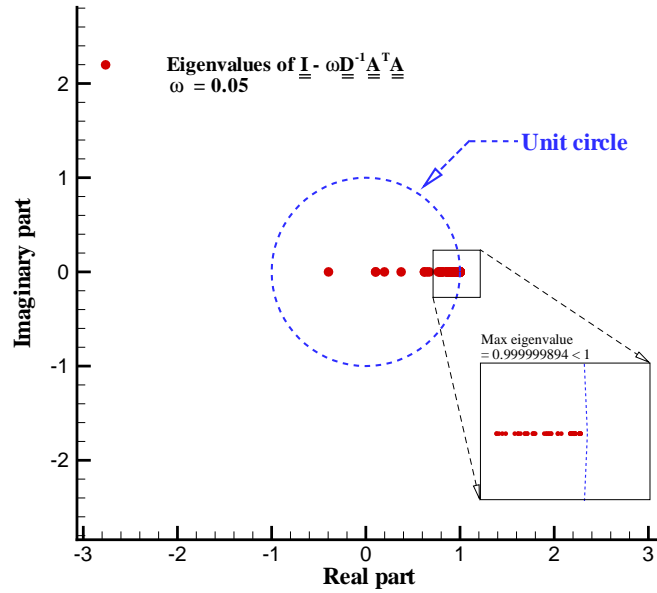


Figure 5.6: Computed eigenvalues for the matrix equation $\underline{\mathbf{I}} - \omega \underline{\mathbf{D}}^{-1} \underline{\mathbf{A}}^T \underline{\mathbf{A}}$

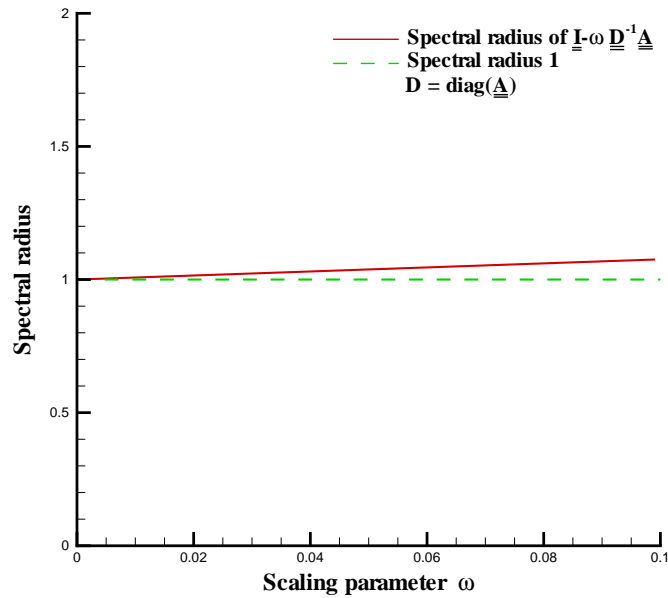


Figure 5.7: Predicted spectral radius of $\underline{\mathbf{I}} - \omega \underline{\mathbf{D}}^{-1} \underline{\mathbf{A}}$ versus the scaling parameter ω .

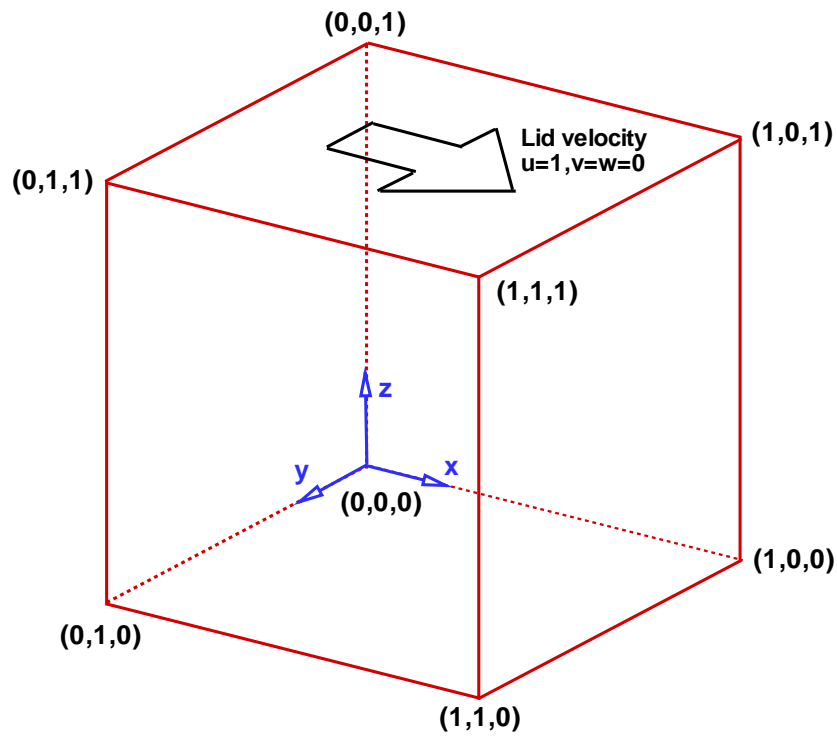


Figure 5.8: Schematic of the three-dimensional lid-driven cavity problem considered in Sec. 5.10.2

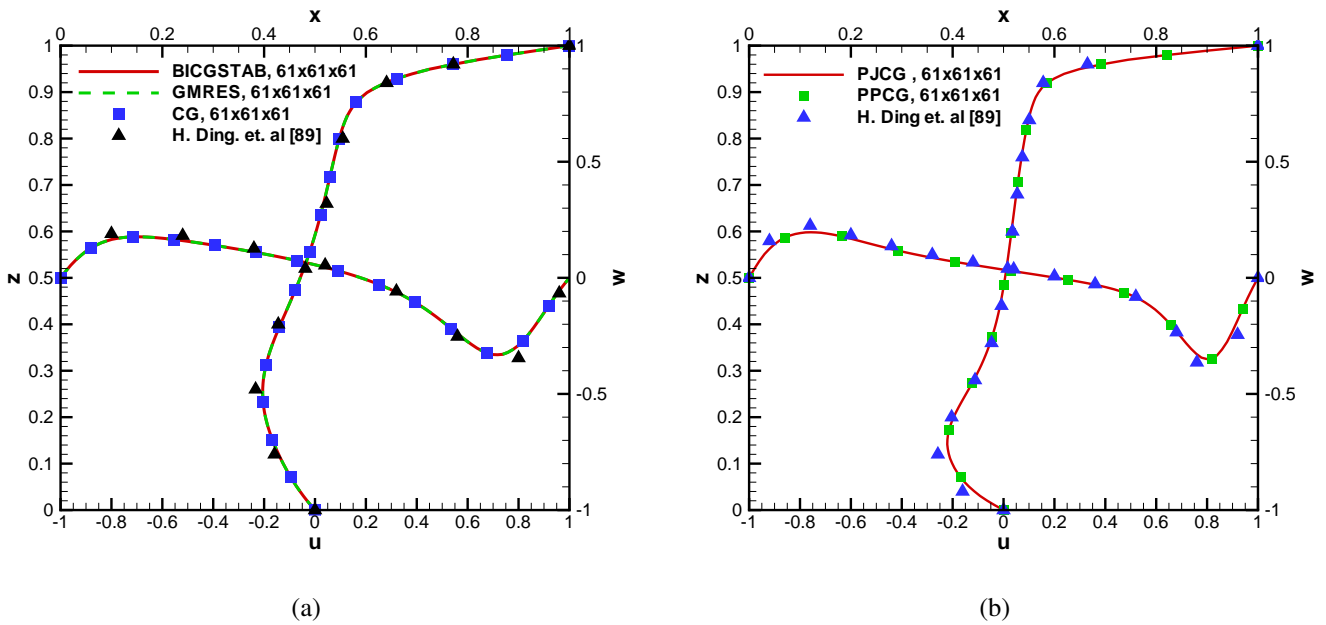
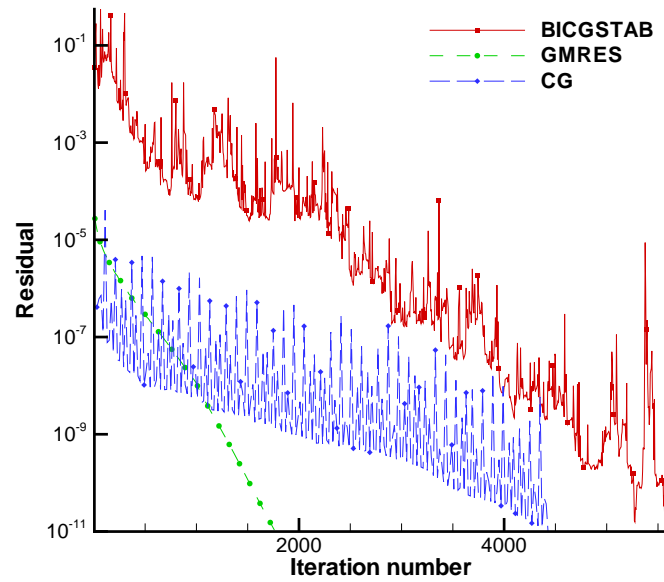
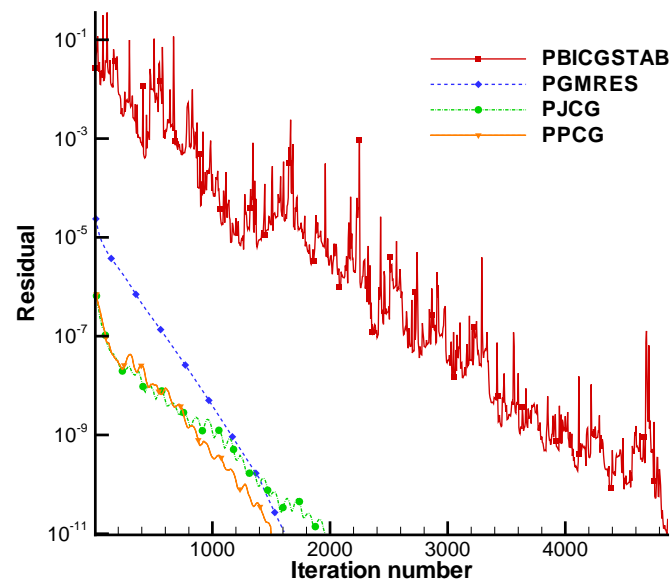


Figure 5.9: Comparison of the predicted velocity profiles at the mid-plane $y = 0.5$. (a) $Re = 400$; (b) $Re = 1000$.



(a)



(b)

Figure 5.10: The residual reduction plots in one inner iteration (at the fourth outer iteration) using the investigated iterative solvers to solve the incompressible Navier-Stokes equations at $Re = 400$ in 41^3 nodal points. (a) Non pre-conditioned solvers; (b) Pre-conditioned solvers.

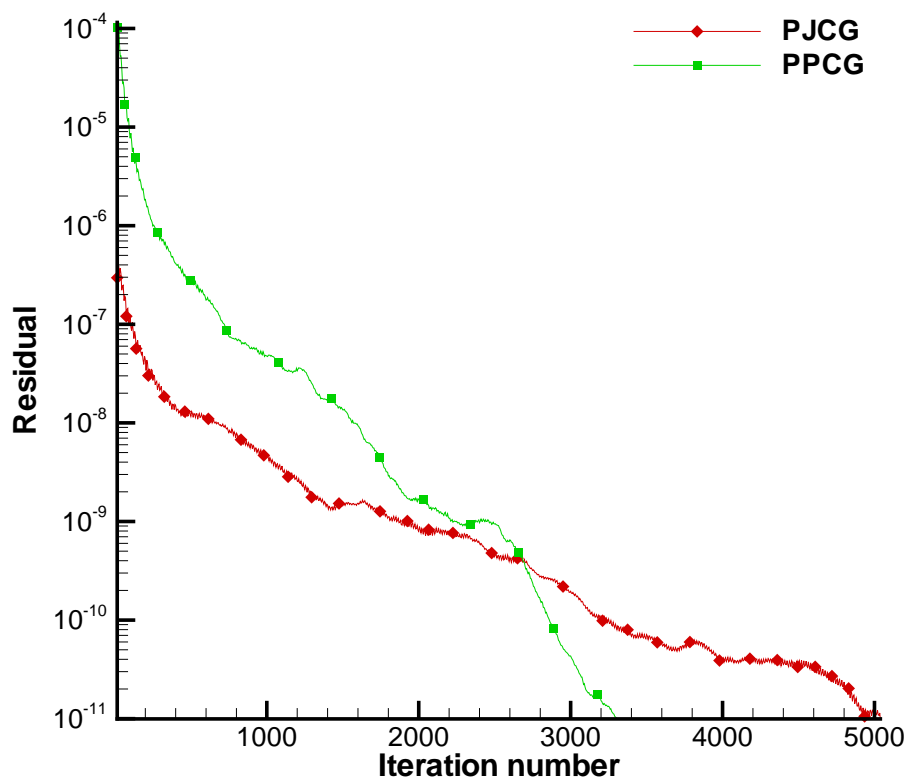


Figure 5.11: The residual reduction plots in one inner iteration (at the fifth outer iteration) using the investigated pre-conditioned iterative solvers to solve the Navier-Stokes equations at $Re = 1000$ in 41^3 nodal points.

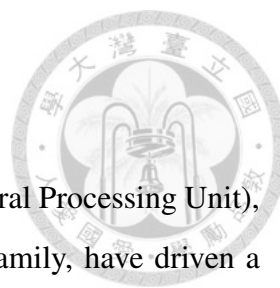


Chapter 6

Parallel computing on GPU

It is well known that the calculation of finite element solution for the incompressible Navier-Stokes equations is a computationally expensive task. In order to reduce the computing time, computers have been developed with many cores executing in parallel (clusters). However, these clusters are too expensive for most of the researchers.

In the past two decades, the Graphic Processing Units (GPUs) have experienced an amazing evolution and become the trend in high performance computing community. From a simple device to generate and exhibit the 2D/3D graphics to a highly parallel, multithread, many-core device with enormous computational power and large memory bandwidth. Today, they have been widely applied to many computational areas due to the characteristic of massive multi-core parallelization, delivery of high throughput on double-precision arithmetic and larger memory bandwidth compared with the traditional CPU processor. This chapter introduces the GPU computing environment and the implementation details of the developed finite element GPU code. Firstly, the short history of GPU and its hardware architecture are introduced. Secondly, the programming model for GPU will be introduced. Due to the fundamental design difference between the CPU and the GPU, some implementation details are introduced in order to obtain the finite element solution on GPU. For the best speedup purpose, some optimization techniques are also introduced. The developed GPU finite element code will be firstly verified by solving the problem amenable to analytical solution. The benchmark lid-driven cavity problem is then solved for the validation and performance analysis study.



6.1 Introduction

In the past decade, the microprocessors based on a single CPU (Central Processing Unit), such as those in the Intel Pentium family and the AMD Opteron family, have driven a rapid performance increase and a dramatic cost reduction in high performance computing. Currently, these microprocessors have reached GFLOP/s and TFLOP/s floating point performance¹ per second to the desktop and the cluster, respectively.

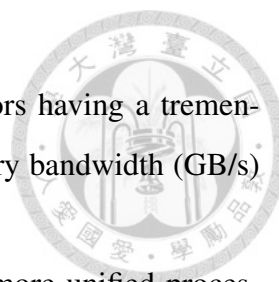
However, the development of CPU became slow since 2003 due to energy consumption and heat dissipation issues that limited the increase of clock rate and the level of procedure activities that can be performed in each clock cycle within a single CPU. To increase the computational power, multiple processing cores were placed on the same chip. This hardware change has exerted a tremendous impact on the software developer community. These multi-core processors are now very useful in general purpose computation but still lack appealing potential for high performance mathematical computations.

The GPUs were, unlike the CPUs, originally designed for rendering high resolution graphics, mainly for the computer game industry. Graphic rendering is in nature highly parallel, involving a large number of arithmetic operations on large data sets of pixels and vertices. The demand for real time, high resolution 2D/3D graphics, has led to an evolution of GPUs into devices with enormous parallel computing capability [90,91]. Several orders of magnitude faster than the fastest multi-core CPU have been realized. Fig. 6.1 depicts the growing gap in peak performance which is measured in FLOP/s between the CPUs and the GPUs over the last decade. It clearly reveals that NVIDIA's GPUs have outperformed Intel CPUs quite substantially.

The memory bandwidth², which represents how fast the memory can be transferred between the CPU/GPU and the DRAM memory, is another important factor which can affect the performance of many computational tasks. Currently, GPUs have operated several times the memory bandwidth of CPUs shown in Fig. 6.2. The above facts reveal

¹Floating point operations per second (FLOPS) is a measure of processor's performance. 1 GFLOP/s = 10^9 FLOP/s, 1 TFLOP/s = 10^{12} FLOP/s

²Bandwidth is an amount of data that can be transferred in a given amount of time

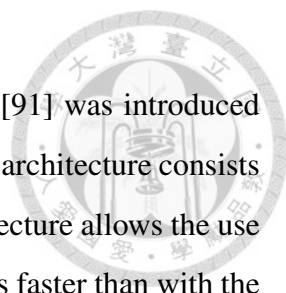


that GPUs are considered to be highly parallel multi-core processors having a tremendous floating point performance peak (GFLOP/s) and higher memory bandwidth (GB/s) compared with the CPU.

While the development of GPU hardware has evolved toward more unified processors, it increasingly resembled high performance parallel computers. Some researchers took notice of its potential computing power for general purpose computing and they started trying to explore the use of GPU to solve the problems in science and engineering communities. The field of GPGPU (General Purpose Graphic Processing Unit) has arisen to be an important topic. The utilization of computational power of the GPU, however, does not come for free. Programmers need to pose their problems as graphic rendering tasks so that computations can be executed on GPUs via open library (e.g. OpenGL) or graphic runtime API (Application Programming Interface) routines. This results in a high learning threshold for programmers who are not familiar with the graphic programming. In addition, the programming environment is very limited when it comes to the stage of debugging

The first breakthrough of GPGPU programming was achieved due to the introduction of Brook [92]. Brook, developed by researchers at Stanford University in 2004, provided the programmers with an extension to C-based programming language that allows accessing GPUs for non-graphical computation. It keeps the programmers away from having a knowledge of graphic programming language, programs written for GPUs using Brook were able to run up to eight times faster than the similar code written for CPUs. Almost at the same time, another similar programming model called Sh, which was developed by researchers at Waterloo University [92], was presented as a library on top of C++. Like Brook, Sh also supports graphic programming and adopts a stream concept for general purpose programming.

In 2006, graphic programming took a leap forward when NVIDIA released its first fully programmable G80-series GPU processor, built on what is called the Tesla architecture, giving the birth of GPGPU computing to a wider public. Later on, a new program-

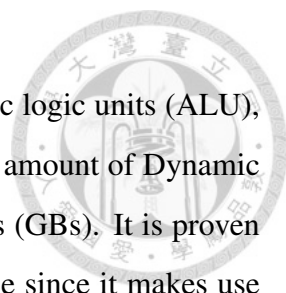


ming model called CUDA (Compute Unified Device Architecture) [91] was introduced by NVIDIA in 2007. CUDA is a general-purpose parallel computing architecture consists of its own programming and execute model. This new parallel architecture allows the use of NVIDIA's GPU to solve many computationally intensive problems faster than with the CPU counterparts. It provides programmers with a set of runtimes API that allow parallel code to be executed only for NVIDIA's GPUs without necessity of learning too much of the graphic programming language. Therefore, GPUs used in non-graphic applications have become an easy task. One of the NVIDIA's goals in the creation of CUDA is to allow programmers with certain background in common programming language (C/C++, Fortran) to be able to access NVIDIA's GPUs as much easy as possible.

The GPUs have found the way into today supercomputers, which increasingly use GPUs for accelerating calculation and reducing power consumption. The newest Top500 list presented in June 2015 showed that three supercomputers on Top10 are composed of CPUs and GPUs. Among them, the TITAN supercomputer in the Oak Ridge National Laboratory is now ranked No. 2 and it was built with both CPUs and GPUs. The increasing use of GPUs in high performance computing industry has made Intel come up with a competing device, the Intel Xeon Phi accelerator [93]. The interest of major developers such as Intel, NVIDIA, and AMD, in the many-core architecture computing industry, promises good future development in the high performance computational area.

6.2 GPU architecture

One natural question to be answered is that "why there is such a large peak-performance gap between CPUs and GPUs?". The answer lies in the difference in the fundamental design targets and the transistors which are distributed differently between these two processors [94]. Fig. 6.3 shows a comparison of the general basic structures of a typical CPU and a typical GPU [94]. In Fig. 6.3, orange region represents transistors devoted to memory, yellow region represents transistors devoted to control logic units, and green region represents transistors which are dedicated to performing arithmetic. Larger re-



gions represent more transistors. The CPU is composed of arithmetic logic units (ALU), control logic units and large cache memories is connected to a large amount of Dynamic Random Access Memory (DRAM), of size in the order of gigabytes (GBs). It is proven that the design of CPU is optimized for sequential code performance since it makes use of sophisticated control logic units to allow instructions from a single thread³ to execute. More importantly, cache⁴ memories are adopted to reduce the instruction, data access latency⁵ of large complex tasks. On the other hand, the GPU is designed to perform pure arithmetic operations. There is no need to devote a large amount of transistors to control unit, branch predictor and cache, etc. Most of the transistors can be, therefore, used for units which perform arithmetic operations. This design makes the GPU be optimized for executing parallel computing tasks. Hereinafter, the term "device" refers to the GPU and the term "host" refers to the CPU will be used.

The revolutionary era of GPU computing started with the G80-series processor by NVIDIA. Later, the GT2000 processor which firstly supports double-precision floating point number that complies with IEEE754 standard [94] was introduced in 2008. Apart from this, a huge amount of data parallelism and data coalesced access is required in order to take advantage of GPU's architecture to achieve high performance. In 2012, the Kepler-series GPU card with GK110 processor was presented. In this dissertation, two different Kepler-series GPU cards, K20 and K40, will be adopted. The detailed features of the adopted Intel CPU, K20 and K40 GPU cards are tabulated in Table. 6.1.

Taking the architecture of Kepler K20 GPU card as an example. Fig. 6.4 shows the basic structure of a CUDA-capable K20 GPU architecture. The K20 GPU has its own DRAM memory that can be up to 5GB to exchange data between the CPU and GPU via the PCI-Express interface. There are different memories in GPU that can be used and controlled by the programmer, from fast to slow memories, depending on the data to be accessed. In order to get a higher performance, some memory access pattern must be

³A thread is an element of data to be processed

⁴Cache is a high-speed memory used to reduce the time to access data from the main memory

⁵Latency can be regarded as the time between a task initialization and the time it begins to execute it.

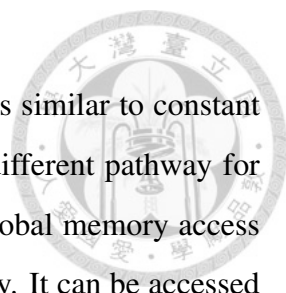
followed. Note that not all algorithms can take advantage of these fast memories.

The GK110 processor in K20 consists of 13 streaming multiprocessors (SMXs). Each SMX is composed of 192 single- and 64 double-precision cores, 4 warp scheduler, 4 dispatch units, 32 special function units (SFUs) responsible for the calculations of some special mathematical functions (e.g. sin, cos, exp, etc). The warp scheduler and the dispatch units enable launch of four warps concurrently. The 32 LD/ST units (Load/Store units) are responsible for a load and store processing. The K20 GPU is clocked at 705 MHz which achieves total single- and double-precision floating point peak performances at 3.52 and 1.17 TFLOP/s, respectively. Each SMX has its own on-chip memory of size 64KB that can be accessed both explicitly as the shared memory and implicitly as the L1 cache. One can refer to NVIDIA Kepler whitepaper [95] for getting more detailed features.

6.2.1 Memory hierarchy

The memory hierarchy in GPU is composed of some different types of memory ranging from a small size with low latency to a large size with large latency. On-chip there are registers, shared memory, and various caches (L1, L2, constant, and texture). Off-chip there are global, local, constant, and texture memories. Local variables defined in device code are stored in registers provided that there are sufficient registers available for use. If there are no sufficient registers, data will be stored in local memory which seriously reduces the performance. Shared memory belongs to the on-chip read/write memory and is accessible only by threads within a block and has a latency of only 1-2 clock cycles. Note that the benefit gained from shared memory can be only obtained if the number of arithmetic operations is larger than the number of memory accesses. The L1 cache consists of programmable shared memory and a general purpose cache. The latter is used to accelerate random access operation. The size of the total L1 cache is 64 KB in Kepler-series GPU.

Constant memory can be accessed and written from the host code, but is accessed only from threads in the device code. It is cache on device and is the most effective when



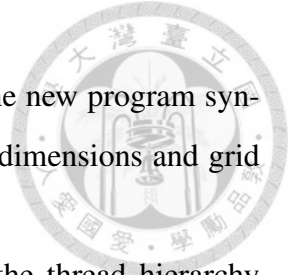
threads execute the same value at the same time. Texture memory is similar to constant memory in that it is accessed-only by device code. It is simply a different pathway for accessing global memory, and is sometimes useful to avoid large global memory access latency. The last level in this hierarchy is the off-chip global memory. It can be accessed by all threads in GPU with the disadvantage of high access latency cost (hundred of clock cycles). One strategy to hide access latency is to increase the parallel working threads. Most of the data are usually stored in global memory since the size of global memory is bigger than other memories. This can be achieved when dealing with huge amount of data. One can refer to Table 6.2 for getting more key features of different device memories.

6.3 CUDA programming model

CUDA is a general-purpose parallel model developed to exploit the computational power of NVIDIA's GPU, based on a scalable programming model on an instruction set architecture. It supports common programming language, enabling programmers to execute a parallel computational task only on NVIDIA's GPU [91]. The idea behind the instruction set architecture is to partition the problem into several smaller problems that can be solved corporately, in parallel, by all threads in GPU.

A significant feature of CUDA model is that it adopts a heterogenous computational framework, meaning that two different types of processors, CPU and GPU, are used together during programming execution. It allows programmers to exploit the strengths of both types of processors. A standard CUDA code can, therefore, be regarded as a standard serial CPU host code plus an additional parallel GPU device code, as illustrated in Fig. 6.5. The host code control function calls for device memory allocation, data transfer between host and device, and function calls. The device code is responsible for time-consuming computational tasks. This heterogeneous programming model simultaneously enables running some codes on the device and some other codes on the host.

The execution of CUDA program is to write and launch the so-called kernel function. A kernel function, which is similar to the subroutine in Fortran, is executed in



parallel on GPU by a set of threads. To launch a kernel function, the new program syntax (`<<<, >>>`) is used and requires launching parameters, block dimensions and grid dimensions.

To simply manage a large amount of threads, CUDA adopts the thread hierarchy framework. The programmers need to decide the number of threads within a block before launching kernel function. When a kernel function is launched, the execution is moved to device and a large number of threads is created. These threads are grouped into blocks, and blocks are grouped into grids. Each block is assigned to a SMX and the threads within a block are executed in a group of 32 threads called warp. The size of warp is fixed and depends on the device. The execution of a warp follows the Single Instruction Multiple Data (SIMD) manner, meaning that each thread within the warp executes the same instruction but acts on different data. This is a major advantage of the SIMD, implying that many instructions and memory operations can be combined and coalesced [94]. The maximum number of threads per block for the Kepler-series GPU in one, two and three-dimensions is 1024 [95].

The threads within the same block can communicate with each other via block synchronization and share data via shared memory. On the other hand, threads from different blocks can not synchronize and can share the data only through the device global memory accompanied with large access latency. Some blocks may be executed in parallel but not necessary all, depending on the available resource. It is required that all blocks should be executed independently. Blocks executing the same kernel function are batched together into a grid. This execution hierarchy is an important feature of CUDA model, which improves dramatically the execution efficiency and makes the GPU scalable.

6.4 CUDA Fortran programming

In the beginning of CUDA development, only the C-based programming language is supported to program and execute parallel codes on GPU. Fortran programmers either rewrite their own developed program code or use `iso_c_binding` interface provided by For-

tran 2003 to call CUDA C kernel function. In the late 2009, the CUDA Fortran compiler became available thanks to the joint work of the Portland Group (PGI) and NVIDIA [96]. Programmers now just need to use a few extensions to Fortran language to modify their code to exploit the power of GPUs in their computations. This section introduces the CUDA Fortran and presents some implementation details.

CUDA Fortran and CUDA C have much in common since CUDA Fortran is based on the CUDA C runtime API. Just as CUDA C is an extension to C language, the CUDA Fortran is also a set of programming extensions to Fortran. However, there are still some differences in how the CUDA concepts are expressed using Fortran programming syntax.

As mentioned before, the execution of CUDA program is to write and launch the kernel function. In CUDA Fortran, the kernel function is defined in Fortran module using the `attribute(global)` qualifier as follows :

```
module CUDA_Fortran_module
  contains
    attributes(global) subroutine kernel_1()
      :
    end subroutine kernel_1
    attributes(global) subroutine kernel_2()
      :
    end subroutine kernel_2
end module CUDA_Fortran_module
```

In the above CUDA Fortran implementation, the attribute `global` indicates that the code is executed on GPU but is called from the host code. To identify the unique index of thread, block and grid, some built-in variables, namely, `gridDim`, `blockDim`, `blockIdx` and `threadIdx` are used. Note that the index of `threadIdx` in CUDA Fortran begins with 1, instead of 0, so a typical index calculation for each thread in CUDA



Fortran should be expressed as

```
i = blockDim%x*(blockidx%x-1)+threadidx%x
```

This is in contrast to CUDA C's :

```
i = blockDim.x*(blockidx.x)+threadidx.x
```

This is the major difference between the CUDA C and CUDA Fortran. The CUDA Fortran also makes use of other attributes such as `shared` or `value`. Data stored in shared memory must be declared in device code with the `shared` variable attributes just as CUDA C uses the `__shared__` qualifier. Some integer/real arguments passing to a kernel function must be declared with the `value` variable attributes.

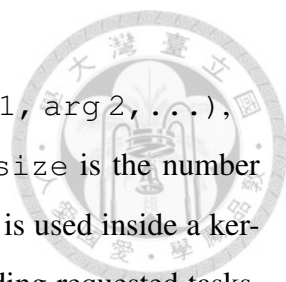
In host code the `CUDAFOR` module, which defines the CUDA runtime API routines, is declared. The device variable in host code is declared with the `device` variable attributes. For example

```
Real(kind=8) , device :: a_d(N)
```

will allocate the one-dimensional array `a_d` of size `N` on device. Device data can also be declared as allocatable using the Fortran `allocate` statement :

```
Real(kind=8), allocatable, device :: a_d(:)
      :
      Ierr = CudaMalloc(a_d(N))
```

In the above, `Ierr` is an integer-type error code and `CudaMalloc()` function will allocate the memory to `a_d` on device. Note that the use of `CudaMalloc()` can guarantee the data alignment in global memory. Once a program has allocated device memory for the data objects the data must be copied from host to device. The data are then copied from device to host after the completion of parallel computation. The data translation is completed via the `CudaMemcpy()` function. One of the two qualifiers `CudaMemcpyHostToDevice` and `CudaMemcpyDeviceToHost` in `CudaMemcpy()` is needed to be specified to indicate the transfer direction. At the end of host code, the `CudaFree()` function is specified to free the memory on the device. Launching the kernel function using the following execution syntax



CALL kernel_name<<< grid_size, block_size >>> (arg 1, arg 2, ...), where grid_size is the number of blocks per grid and block_size is the number of threads per block. Use of the Syncthreads () function which is used inside a kernel function is to guarantee that device has completed all the preceding requested tasks. Reader can refer to the programming guide [96, 97] for more detailed implementation. Global synchronization is not allowed by the CUDA Fortran model. The only way to force a global synchronization is to exit the kernel function before launching a new one.

6.5 Implementation of iterative solvers on GPU

One of the applications that might be benefitted from the enormous computational power of modern GPU is the application of iterative solvers to solve a large system of linear equations. The main objective of this chapter is to solve the finite element equations discretized from the incompressible Navier-Stokes equations on GPU. Special attention was devoted to the sparse matrix-vector product (SpMV) operation due to its importance in applying an iterative solver [98–100]. A good summary on the development of high performance SpMV implementation on recent multicore and accelerator-based hardware can be found in [101]. Some iterative solvers implemented on GPU architecture have been reported in [102–104].

In Chapter 5, the efficiency of using the pre-conditioned conjugate gradient (PCG) solver for solving the normalization matrix equations has been demonstrated. Now, the current PCG solver will be implemented on GPU platform to obtain convergent solution even quickly. The PCG algorithm is described as follows :

- Starting from an initial guess $\underline{\mathbf{x}}_0$
- $\underline{\mathbf{M}}$ represents the Jacobi pre-conditioner
- Compute $\underline{\mathbf{x}}'_0 = \sum_e \underline{\mathbf{A}}_e \underline{\mathbf{x}}_0$, $\underline{\mathbf{x}}''_0 = \sum_e (\underline{\mathbf{A}}_e)^T \underline{\mathbf{x}}'_0$, $\underline{\mathbf{b}}' = \sum_e (\underline{\mathbf{A}}_e)^T \underline{\mathbf{b}}$ // matrix-vector product ;
- Compute the initial residual $\underline{\mathbf{r}}_0 = \underline{\mathbf{b}}' - \underline{\mathbf{x}}''_0$ // global update ;
- Solve $\underline{\mathbf{M}} \underline{\mathbf{z}}_0 = \underline{\mathbf{r}}_0$ // pre-conditioning ;



$$\underline{\mathbf{p}}_0 = \underline{\mathbf{z}}_0$$

For $j = 1, 2, \dots,$

$$\underline{\mathbf{p}}'_{j-1} = \underline{\mathbf{A}}\underline{\mathbf{e}}\underline{\mathbf{p}}_{j-1} \quad // \text{matrix-vector product ;}$$

$$\underline{\mathbf{p}}''_{j-1} = \underline{\mathbf{A}}^T \underline{\mathbf{p}}'_{j-1} \quad // \text{matrix-vector product ;}$$

$$\alpha_{j-1} = (\underline{\mathbf{p}}_{j-1}, \underline{\mathbf{r}}_{j-1}) / (\underline{\mathbf{p}}_{j-1}, \underline{\mathbf{p}}''_{j-1}) \quad // \text{inner product ;}$$

$$\underline{\mathbf{x}}_j = \underline{\mathbf{x}}_{j-1} + \alpha_{j-1} \underline{\mathbf{p}}_{j-1} \quad // \text{global update ;}$$

$$\underline{\mathbf{r}}_j = \underline{\mathbf{r}}_{j-1} - \alpha_{j-1} \underline{\mathbf{p}}''_{j-1} \quad // \text{global update ;}$$

Convergence check

$$\text{Solve } \underline{\mathbf{M}}\underline{\mathbf{z}}_j = \underline{\mathbf{r}}_j \quad // \text{pre-conditioning ;}$$

$$\beta_{j-1} = (\underline{\mathbf{z}}_j, \underline{\mathbf{r}}_j) / (\underline{\mathbf{r}}_{j-1}, \underline{\mathbf{r}}_{j-1}) \quad // \text{inner product ;}$$

$$\underline{\mathbf{p}}_j = \underline{\mathbf{z}}_j + \beta_{j-1} \underline{\mathbf{p}}_{j-1} \quad // \text{global upadte ;}$$

End

The main operations of PCG solver consist of (i) global update operation ; (ii) pre-conditioning operation ; (iii) inner-product operation ; and (iv) matrix-vector product operation. The main computational cost is the matrix-vector product operation (iV), in particular in large-sized problem, while the cost of other operations is relatively small. The operations (i) and (ii) are naturally parallelized owing to the chosen Jacobi preconditioner. The implementations of operations (iii) and (iv) will be described in detail.

6.5.1 Inner product operation on GPU

Calculation of the inner product operation (iii) on GPU is implemented in two steps. Firstly, the temporal vector $\underline{\mathbf{c}}$ with size N is constructed to store the vector-vector product $c_i = a_i b_i$ ($i = 1, \dots, N$) of vectors $\underline{\mathbf{a}}$ and $\underline{\mathbf{b}}$ in GPU. Secondly, all the components in $\underline{\mathbf{c}}$ are summed up to get the inner product $(\underline{\mathbf{a}}, \underline{\mathbf{b}}) = \sum_{i=1}^N c_i$. The above procedures can be completed with two kernel functions: One is used for the vector-vector product and the other one is used for the calculation of the sum.



6.5.2 Matrix-vector product operation on GPU

It is proven that the SpMV operation is regarded as the most computationally intensive part in most of the Krylov subspace iterative solvers. Great efforts have been made to improve the performance of parallel SpMV implementations. Several research groups have reported their implementations on high-performance SpMV on GPU [98–100]. A good summary on the development of high performance SpMV implementation on recent multi-core and accelerator-based hardware can be found in [101]. In these literatures, the global matrix is, however, usually stored in some specific sparse matrix formats. The choice of sparse matrix format essentially depends on the feature of the matrix and the number of non-zero entries. The survey of sparse matrix can be found in the literature [72]. Among them, the CSR format is the most common one and has been frequently used.

As mentioned already, the use of sparse matrix format still encounters memory intensive requirement problem because the number of non-zero entries increases significantly as the problem size increases. In this study, we aimed to compute the matrix-vector product without using any sparse matrix format. To this end, the EBE technique will be implemented on GPU. Kiss et al. [105] also presented the similar technique, but the detailed algorithm is not given therein.

In EBE context, the global matrix-vector $\underline{\underline{\mathbf{A}}}\underline{\underline{\mathbf{x}}}$ can be decomposed into the sum of the element-level matrix-vector (EMV) $\underline{\underline{\mathbf{A}}}^e \underline{\underline{\mathbf{x}}}^e$

$$\underline{\underline{\mathbf{A}}}\underline{\underline{\mathbf{x}}} = \sum_{e=1}^{Nel} (\underline{\underline{\mathbf{B}}}^e)^T (\underline{\underline{\mathbf{A}}}^e \underline{\underline{\mathbf{x}}}^e) \quad (6.1)$$

where $\underline{\underline{\mathbf{B}}}^e$ denotes the Boolean matrix, which maps the entries of e-th element matrix $\underline{\underline{\mathbf{A}}}^e$ into a global matrix $\underline{\underline{\mathbf{A}}}$. Calculation of the global matrix-vector product on GPU via Eq. (6.1) is, therefore, a challenging task.

In the developed EMV kernel function (Algorithm 6.1), each block with n_e threads (n_e being the local degree of freedom) is responsible for computing the product of one row vector of matrix to its corresponding vector. Therefore, n_e blocks are used to compute



one element matrix-vector product. The above algorithm is described in Algorithm 6.1

Algorithm 6.1: EMV kernel function

```

1  $\underline{\mathbf{A}}^e(n_e, n_e)$  : The e-th elementary matrix element matrix ;
2  $\underline{\mathbf{x}}^e(n_e)$  : corresponding vector ;
3  $\mathbf{B}_i$  : The i-th block containing  $n_e$  threads in device ;
4  $\mathcal{E}$  : A collection of elements ;
5 for  $e \in \mathcal{E}$  do
6   for  $i = 1 \rightarrow n_e$  do
7     Compute  $\overbrace{\underline{\mathbf{A}}_{i,j}^e \underline{\mathbf{x}}_j^e}^{\text{partial product}}$  and store in  $\mathbf{B}_i$  //parallel ;
8     Compute the sum  $\sum_{j=1}^{n_e} (\underline{\mathbf{A}}_{i,j}^e \underline{\mathbf{x}}_j^e)$  //parallel ;
9   end
10 end

```

The Algorithm 6.1 may, however, result in an incorrect result due to the presence of the so-called race condition [105]. Race condition occurs in GPU computation if any two threads try to read or write the same GPU memory location simultaneously. Such a read or write leads to information loss and incorrect result. Since in the CUDA execution no information is given about which thread performs the specific task, the fastest thread will perform the task.

To avoid race condition problem, one can use the atomic update or the element coloring technique [105]. The former refers to protect the memory space during I/O causing other threads to access the same memory space to wait until the operation is fully completed. The latter refers to partition elements into a finite disjoint element subsets marked with different colors such that any two elements in a given subset are not allowed to share the same global node, as illustrated in Figs. 6.6-6.7. Different colors are displaced in process serially, while elements within the subset marked with the same color are invoked in parallel. In this study, the element coloring strategy shown in Algorithm 6.2 is adopted because it is effective and run only once during the 2D/3D mesh generation.

The Algorithm 6.2 is very simple, but it may generate color distributions that are usually uneven. We therefore add a second step to balance the number of elements inside



Algorithm 6.2: 2D/3D Mesh coloring algorithm

```
1  $Nel$       : The number of elements ;
2  $\mathcal{E}$       : A collection of elements ;
3  $\mathcal{E}^e$      : The e-th element ;
4  $Color(Nel)$  : The color ID of each element ;
5 for  $e=1 \rightarrow Nel$  do
6    $Cused(1) = 1, \dots, Cused(Nel)=Nel$  // Initialize the Cused array ;
7   for  $k = 1 \rightarrow Nel-1$  do
8     check the all neighbors of e-th element  $\mathcal{E}^k \in \mathcal{E}/\mathcal{E}^e$  ;
9     if ( $\mathcal{E}^k$  is colored) then
10       $Cused(Color(k)) = Nel+1$ 
11    end
12  end
13   $Color(e) = \text{Min}\{Cused\}$  // mark the element with color ;
14 end
```

each mesh subset of a given color, as stated in [106].

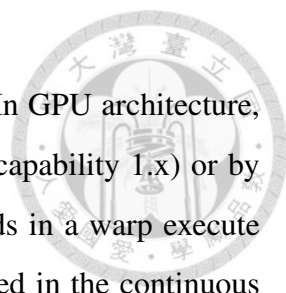
6.6 Optimization

In order to maximize the GPU's computation power for the finite element calculation, the currently developed device code must be optimized via several aspects of the kernel function design. In this section, two techniques are adopted in order to improve the performance of speedup. The first skill is to exploit a global memory coalescing. The other one is related to the utilization of the shared memory. Both skills are described below.

6.6.1 Global memory coalescing

Before launching the kernel functions on GPU, data must be copied from host to device and store in the global memory. Global memory is a large-sized memory but has a much higher latency (about 400 to 800 clock cycles). It is not cached, so it is very important to follow a right pattern to access global memory efficiently.

Global memory is typically accessed via DRAM (Dynamic Random Access Memory). Accessing DRAM is normally a slow process and has thus a large access latency and a finite access bandwidth. How to optimally access data from global memory plays a key



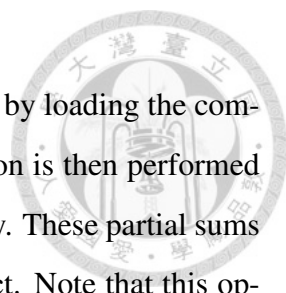
role to determine if a better parallel performance can be achieved. In GPU architecture, global memory is accessed by a half warp (for device of compute capability 1.x) or by a warp (for device of compute capability 2.x). When all the threads in a warp execute the load instruction, the hardware will detect what has been accessed in the continuous memory region. If all threads in a warp access the continuous global memory region, then these accesses can be coalesced and combined into a single memory transaction. Therefore, storing the data in a special aligned data format to guarantee that the threads in a warp can access the continuous global memory region is worthy to be done.

To avoid race condition, elements are divided into several different disjoint subsets and they are marked by different colors. Our strategy to improve the access efficiency is to store all the elementary matrices $\underline{\underline{A}}^e$ in global memory in the color order instead of the element id order, as illustrated in Fig. 6.8. The advantage of this data arrangement is that the global memory coalescing can be guaranteed. In addition, the degree of freedom (DOF) for each element is 22 in two- and 89 in three-dimensional problem in the chosen velocity-pressure interpolation functions. In the EMV kernel function, the block size is, therefore, chosen as the DOF in order to compute the SpMV product exactly. However, it is known that the number of threads has been chosen as the multiple of 32 which is the size of a warp to render a high performance [94]. One can therefore extend the length of each row from the DOF to the multiple of warp size (e.g. 96 in three-dimensional problem) by filling up the zero entries, as illustrated in Fig. 6.9.

6.6.2 Shared memory

Shared memory is classified to be the on-chip read/write memory and threads within the same block can access and share the data with the latency of only 1-2 clock rates. It is therefore considered to be an important optimization technique and it can provide us an efficient means for threads to cooperate with each other.

In PCG solver, an important question in inner product operation (iii) is how the temporal products in vector \underline{c} are summed up effectively on GPU. The vector \underline{c} is originally stored in the global memory. To improve the computational efficiency, each thread block



can calculate a partial sum of a section of vector \underline{c} . This can be done by loading the components of the section into the shared memory. The parallel reduction is then performed in each block to obtain the partial sum quickly via the shared memory. These partial sums are then copied from device to host to sum up the final inner product. Note that this optimization technique is also applied to Algorithm 6.1 (line 8) with the block size of n_e threads.

6.7 Description of GPU code

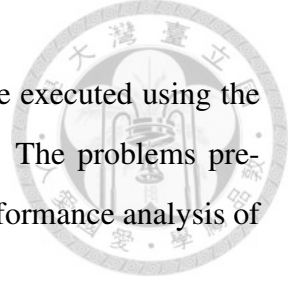
The flow chart of the developed finite element GPU code for solving the fluid and heat flow problems is stated in Fig. 6.11, The mesh is generated by using the house-developed program for regular geometry and commercial mesh generator Gambit for irregular geometry. Irregular geometry is constructed by using the commercial software Rhino.

In order to run the program, two input files are required. One is the mesh file containing the basic mesh parameters, domain coordinates, mesh connectivity, fluid properties and boundary conditions. Another is the data of the inlet and outlet boundary when solving the inlet/outlet flow problem.

When executing the program, the problem loads the mesh file and organizes some necessary data. These data are then copied from host to device for finite element calculations. The GPU takes the most expensive computational tasks. The CPU is responsible for organizing the data, calling the kernel functions and checking the convergence. Calculation will stop if the convergence criterion is satisfied. All the computations were performed with an Intel I7-4820K (CPU) containing 8 cores. Since the problem size is limited by the on-board GPU global memory, the two- and three-dimensional computations were performed on K20 and K40 GPU, respectively.

6.8 Numerical study

Two test problems, including the analytical verification and the benchmark lid-driven cavity flow problems, are chosen to test the currently developed finite element GPU code for



the incompressible Navier-Stokes equations. All the calculations are executed using the IEEE754 satisfying double-precision floating-point representation. The problems presented here will be also solved using the serial CPU code for the performance analysis of speed up.

6.8.1 2D verification study

The first step toward verification of the GPU code is to solve equations amenable to analytical solutions. In the unit square domain the problem amenable to the analytical solutions is given below

$$u = 1 - \exp^{\lambda x} \cos(2\pi y) \quad (6.2)$$

$$v = \frac{\lambda}{2\pi} \exp^{\lambda x} \sin(2\pi y) \quad (6.3)$$

$$p = \frac{1}{2}(1 - \exp^{2\lambda x}) \quad (6.4)$$

where $\lambda = \frac{Re}{2} - (\frac{Re^2}{4} + 4\pi^2)^{1/2}$. The Reynolds number chosen for this study is 1000. The predicted L_2 error norms obtained at different nodal points are tabulated in Table. 6.3. Good agreement between the numerical and analytical solutions is clearly shown. For the sake of comparison, all the calculations will be also carried out using the serial CPU code and the predicted L_2 norms are tabulated in Table. 6.3 as well.

6.8.2 3D verification study

The problem under investigation is defined in a hexahedron of length 1. Along the cube surfaces, the nodal velocities are analytically given below.

$$u = \frac{1}{2}(y^2 + z^2) \quad (6.5)$$

$$v = -z \quad (6.6)$$

$$w = y \quad (6.7)$$

The corresponding analytical pressure takes the form $p = \frac{1}{2}(y^2 + z^2) + \frac{2}{Re}x$. All the calculations at $Re = 1000$ are carried out using the serial CPU and parallel GPU code, respectively. The predicted L_2 error norms tabulated in Table 6.4 show good agreement



with the analytical solutions. The developed finite element GPU code for solving the two- and three-dimensional incompressible Navier-Stokes equations is analytically verified.

6.8.3 Lid-driven cavity flow problem

The benchmark lid-driven cavity flow problems schematic shown in Fig. 4.1 and Fig. 5.8 are used to validate the developed finite element GPU code. In a unit cubical cavity, a unit velocity is prescribed on the lid boundary while on the other boundary no-slip condition $\underline{u} = 0$ is specified. The predicted two- and three-dimensional velocity profiles at different Reynolds numbers are compared with the benchmark solutions of Ghia [52] and Ding [89]. Good agreement shown in Fig. 6.12. demonstrates that the developed finite element GPU code can be used to accurately predict the incompressible fluid flow in the cavity.

6.9 Performance analysis

6.9.1 Introduction

In this section, a performance comparison between the developed CPU and GPU code is discussed, with the goal of exploring the benefit of running the computer program on GPU. The benchmark lid-driven cavity problem investigated before is used to perform computations at different grid sizes to assess the speedup performance.

Since GPU is suitable for parallel algorithm, especially for those with high data parallelism. The performance analysis in this section is based on the degree of grids refinement. By increasing the grid sizes, the amount of load and computing time also increases significantly. In order to analyze the performance, a summary of some important routines in the current developed code is listed in Table 6.5. The bold red triangle symbol indicates that the routine is fully executed on CPU, whereas the green circle symbol indicates that the routine is fully executed on GPU. Note that the hybrid CPU/GPU platform requires three additional routines in comparison with the CPU platform counterpart. This table is further used to address the time measurement obtained from both CPU and GPU codes.



Following the nomenclature given to each routine, a short description of some routines is given below.

- **Data preparation**

This routine consists of loading the domain coordinate, mesh connectivity information and boundary condition data. In addition, the values of interpolation functions, derivative of interpolation functions and weighted factors associated with the total Gaussian quadrature points are also evaluated. Moreover, the Boolean matrices for each element are constructed in this routine.

- **Element coloring**

The coloring routine (Algorithm 6.2) runs on CPU since it is effective and runs only once. Note that this routine is only executed on a hybrid CPU/GPU platform.

- **Initial condition**

The steady-state solutions at low Reynolds number are used to be the initial condition for all the problems at high Reynolds number.

Time measurements of each routine for the CPU and GPU code are obtained via the command `clock_system()`. Note that the command `Syncthreads()` in GPU code must precede the time measurement command in order to guarantee all the threads in GPU have completed all computations.

The two- and three-dimensional lid-driven cavity flow problems at two different Reynolds number are used to assess the performance of CPU and CPU/GPU platforms. The details of the grid sizes used in the performance analysis are listed in Tables. 6.6 and 6.7. The nomenclature introduced in Table 6.5 will be useful to clearly indicate the most time-consuming routine and how they scale with mesh refinement.

The first task toward the performance analysis is to justify the need of GPU platform. For all the considered cases, the computation time and its relative time of each routine executed on CPU platform are tabulated in Tables. 6.9.1-6.9.1. One can clearly see that the

SOLV routine is the most time-consuming, up to 99% of the total computation time. The computation time for executing other routines is relatively small and can be negligible.

It is well known that the good parallel performance is achieved if the most computationally intensive part is accelerated. This justifies the need for a complete implementation of the iterative solver on GPU in order to take advantage of the available GPU computing power. The computation time of each routine executed on a hybrid CPU/GPU platform is tabulated in Tables. 6.12-6.15 and the speedup ratio between the these two platforms are summarized in Table. 6.16- 6.19 by presenting the total computational time and the respective speedup ratio. The speedup indicates how fast the GPU runs in comparison with its CPU counterpart. The speedup ratio for all considered cases are plotted in Figs. 6.13. The speedup increases with respect to the increasing grid sizes and Reynolds number. Thus the increase of computation tasks raises the speedup ratio considerably higher.

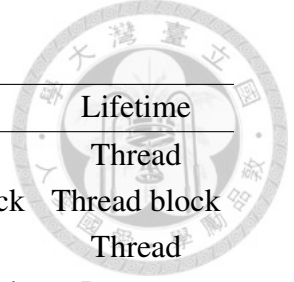
	CPU	GPU 1	GPU 2
Processor	Intel i7-4820K	NVIDIA Kepler K20	NVIDIA Kepler K40
Clock rate	3.7 GHz	705 MHz	745 MHz
Number of cores	8	2496 (SP) 832 (DP)	2880 (SP) 960 (DP)
off-chip memory	32 GB (DDR3)	5 GB (DDR5)	12 GB (DDR5)
on-chip memory	L1-cache: 256 KB/core L2-cache: 1024 KB/core L3-cache: 10 MB/core	Constant memory: 64 KB Shared memory: 48 KB/Block Register : 64 KB/Block	Constant memory: 64 KB Shared memory: 48 KB/Block Register : 64 KB/Block
Peak flops	59.2 GB/s (DP)	3.52 TB/s (SP) 1.17 TB/s (DP)	4.29 TB/s (SP) 1.43 TB/s (DP)
Memory bandwidth	59.7 GB/s	208 GB/s	288 GB/s
IEEE754 single/double	yes / yes	yes / yes	yes / yes

SP : single precision

DP : double precision

Table 6.1: Some key specifications of the considered CPU processor, and the NVIDIA K20 and K40 GPU cards





Memory	Location	Cached	Device access	Scope	Lifetime
Register	On-chip	N/A	R/W	One thread	Thread
Shared	On-chip	N/A	R/W	All threads in block	Thread block
Local	DRAM	Yes	R/W	One thread	Thread
Global	DRAM	Yes	R/W	All threads in block	Program
Constant	DRAM	Yes	R	All threads in block	Program
Texture	DRAM	Yes	R	All threads in block	Program

Table 6.2: Some features of different GPU device memory types

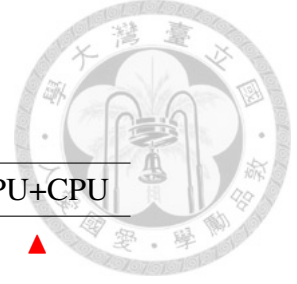
Grid size	Platform	$\ \mathbf{u} - \mathbf{u}_{exact}\ _2$	$\ \mathbf{v} - \mathbf{v}_{exact}\ _2$	$\ \mathbf{p} - \mathbf{p}_{exact}\ _2$
21^2	CPU	6.182×10^{-5}	5.708×10^{-6}	2.785×10^{-5}
	Hybrid	6.182×10^{-5}	5.708×10^{-6}	2.785×10^{-5}
41^2	CPU	5.656×10^{-6}	4.392×10^{-7}	4.464×10^{-6}
	Hybrid	5.656×10^{-6}	4.392×10^{-7}	4.464×10^{-6}
61^2	CPU	1.630×10^{-6}	1.494×10^{-7}	1.949×10^{-6}
	Hybrid	1.630×10^{-6}	1.495×10^{-7}	1.950×10^{-6}
81^2	CPU	7.183×10^{-7}	9.592×10^{-8}	1.189×10^{-6}
	Hybrid	7.183×10^{-7}	9.591×10^{-8}	1.189×10^{-6}

Note : Hybrid denotes the hybrid CPU/GPU platform

Table 6.3: The computed L_2 error norms obtained at different grids for the 2D verification problem considered in Sec. 6.8.1.

Grid size	Platform	$\ \mathbf{u} - \mathbf{u}_{exact}\ _2$	$\ \mathbf{v} - \mathbf{v}_{exact}\ _2$	$\ \mathbf{w} - \mathbf{w}_{exact}\ _2$	$\ \mathbf{p} - \mathbf{p}_{exact}\ _2$
21^3	CPU	2.450×10^{-4}	1.739×10^{-3}	1.998×10^{-3}	9.945×10^{-3}
	Hybrid	2.450×10^{-4}	1.739×10^{-3}	1.997×10^{-3}	9.943×10^{-3}
41^3	CPU	1.451×10^{-4}	9.783×10^{-4}	1.170×10^{-3}	4.936×10^{-3}
	Hybrid	1.451×10^{-4}	9.783×10^{-4}	1.170×10^{-3}	4.935×10^{-3}
61^3	CPU	1.033×10^{-4}	6.791×10^{-4}	8.248×10^{-4}	3.305×10^{-3}
	Hybrid	1.032×10^{-4}	6.791×10^{-4}	8.247×10^{-4}	3.303×10^{-3}
81^3	CPU	8.000×10^{-5}	5.196×10^{-4}	6.354×10^{-4}	2.480×10^{-3}
	Hybrid	8.000×10^{-5}	5.196×10^{-4}	6.365×10^{-4}	2.481×10^{-3}

Table 6.4: The computed L_2 error norms obtained at different grids for the 3D verification problem considered in Sec. 6.8.2.



Routines	Nomenclature	CPU	GPU+CPU
Data preparation	DATA	▲	▲
Elements coloring	ECOL	–	▲
Host to device	HTOD	–	▲
Compute all $\underline{\underline{\mathbf{A}}}$ ^e	MATX	▲	▲
Compute $\underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{b}}}$	NORB	▲	●
Compute $\underline{\underline{\mathbf{M}}}$	PREC	▲	●
Solve $\underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{A}}}^T \underline{\underline{\mathbf{b}}}$	SOLV	▲	●
Device to host	DTOH	–	▲

$\underline{\underline{\mathbf{M}}}$: Jacobi pre-conditioner

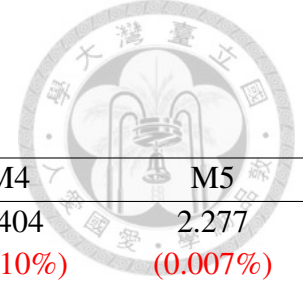
Table 6.5: Summary of the most important routines on the implementation platforms

	M1	M2	M3	M4	M5
Element sizes	1600	2500	3600	6400	10000
Node sizes	6561	10201	14641	25921	40401
Total DOF	14803	23003	33003	58403	91003

Table 6.6: Meshes used in the performance analysis of the 2D lid-driven cavity problem considered in Sec. 6.8.3.

	M1	M2	M3	M4	M5
Element sizes	1000	3375	8000	27000	64000
Nodal sizes	9261	29791	68921	226981	531441
Total DOF	29114	93469	216024	710734	1663244

Table 6.7: Meshes used in the performance analysis of the 3D lid-driven cavity problem considered in Sec. 6.8.3.

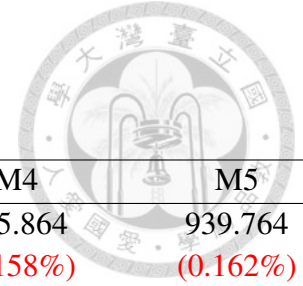


	M1	M2	M3	M4	M5
DATA	0.304 (0.021%)	0.475 (0.016%)	0.868 (0.016%)	1.404 (0.010%)	2.277 (0.007%)
MATX	2.037 (0.143%)	2.925 (0.101%)	4.086 (0.076%)	7.330 (0.052%)	11.241 (0.035%)
NORB	0.245 (0.017%)	0.378 (0.013%)	0.487 (0.009%)	0.892 (0.006%)	1.349 (0.004%)
PREC	0.407 (0.029%)	0.609 (0.021%)	0.814 (0.015%)	1.499 (0.011%)	2.355 (0.007%)
SOLV	1362.064 (99.781%)	2845.961 (99.846%)	5313.823 (99.882%)	14162.614 (99.922%)	32029.803 (99.772%)
Total	1365.057	2850.348	5320.078	14173.739	32047.025

Table 6.8: Timing in seconds of each routine executed on CPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 1000$. The value is "(%)” denotes the relative time.

	M1	M2	M3	M4	M5
DATA	0.302 (0.007%)	0.472 (0.006%)	0.689 (0.005%)	1.248 (0.004%)	1.914 (0.003%)
MATX	2.813 (0.070%)	3.108 (0.038%)	4.410 (0.031%)	8.213 (0.024%)	12.421 (0.019%)
NORB	0.319 (0.008%)	0.381 (0.005%)	0.531 (0.004%)	1.031 (0.003%)	1.519 (0.002%)
PREC	0.508 (0.013%)	0.657 (0.008%)	0.914 (0.006%)	1.541 (0.005%)	2.623 (0.004%)
SOLV	4025.668 (99.902%)	8245.954 (99.944%)	14120.571 (99.954%)	34015.611 (99.965%)	66933.028 (99.972%)
Total	4029.610	8250.572	14127.115	34027.644	66951.505

Table 6.9: Timing in seconds of each routine executed on CPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 5000$. The value is "(%)” denotes the relative time.



	M1	M2	M3	M4	M5
DATA	4.227 (0.179%)	17.721 (0.184%)	49.545 (0.167%)	245.864 (0.158%)	939.764 (0.162%)
MATX	12.027 (0.508%)	36.909 (0.384%)	94.396 (0.317%)	554.422 (0.355%)	1862.125 (0.321%)
NORB	0.452 (0.019%)	1.310 (0.014%)	3.104 (0.010%)	17.117 (0.011%)	69.612 (0.012%)
PREC	1.918 (0.081%)	5.772 (0.060%)	13.915 (0.047%)	101.148 (0.065%)	307.454 (0.053%)
SOLV	2347.659 (99.213%)	9544.219 (99.358%)	29570.941 (99.459%)	154695.416 (99.411%)	576922.149 (99.452%)
Total	2366.283	9605.931	29731.901	155611.971	580101.103

Table 6.10: Timing in seconds of each routine executed on CPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 400$. The value is ”()” denotes the relative time.

	M1	M2	M3	M4	M5
DATA	4.118 (0.049%)	17.706 (0.066%)	49.514 (0.062%)	343.546 (0.063%)	885.999 (0.061%)
MATX	11.996 (0.142%)	36.551 (0.136%)	82.742 (0.104%)	577.879 (0.106%)	1568.655 (0.108%)
NORB	0.374 (0.004%)	1.029 (0.004%)	2.745 (0.004%)	27.258 (0.005%)	87.148 (0.006%)
PREC	1.918 (0.023%)	5.070 (0.019%)	12.027 (0.015%)	70.872 (0.013%)	334.066 (0.023%)
SOLV	8403.321 (99.781%)	26834.044 (99.776%)	79235.325 (99.815%)	544148.948 (99.813%)	1449582.869 (99.802%)
Total	8421.727	26894.400	79382.353	545168.143	1452458.737

Table 6.11: Timing in seconds of each routine executed on CPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 1000$. The value is ”()” denotes the relative time.



	M1	M2	M3	M4	M5
DATA	0.321	0.505	0.716	1.298	1.989
ECOL	0.217	0.373	0.609	0.921	1.879
HTOD	0.246	0.385	0.533	0.929	1.424
MATX	2.037	3.509	4.541	8.128	13.039
NORB	0.054	0.074	0.093	0.161	0.239
PREC	0.797	1.628	1.809	3.158	4.966
SOLV	253.187	508.984	895.346	3254.107	5317.635
DTOH	0.005	0.008	0.008	0.015	0.022
Total	256.864	515.466	903.655	3268.717	5341.193

Table 6.12: Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 1000$

	M1	M2	M3	M4	M5
DATA	0.333	0.569	0.873	1.334	2.004
ECOL	0.217	0.373	0.609	0.921	1.879
HTOD	0.266	0.420	0.581	1.014	1.593
MATX	2.318	3.441	5.151	9.173	14.544
NORB	0.051	0.082	0.101	0.169	0.254
PREC	0.904	1.352	2.041	3.528	5.540
SOLV	754.137	1717.714	2410.355	5553.297	10965.770
DTOH	0.002	0.004	0.011	0.019	0.031
Total	758.228	1723.955	2419.722	5569.455	10991.615

Table 6.13: Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 2D lid-driven cavity flow problem investigated at $Re = 5000$

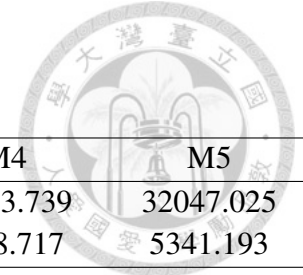


	M1	M2	M3	M4	M5
DATA	11.882	31.178	92.289	550.847	2303.337
ECOL	1.232	13.587	76.736	837.616	4566.336
HTOD	1.729	5.620	7.703	43.606	70.603
MATX	22.585	75.138	167.243	625.483	1569.951
NORB	0.005	0.014	0.025	0.072	0.176
PREC	2.804	11.469	24.521	82.362	193.155
SOLV	176.580	700.007	2130.097	10348.068	33337.453
DTOH	0.002	0.007	0.023	0.070	0.110
Total	216.819	837.020	3498.637	11650.508	37474.785

Table 6.14: Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 400$

	M1	M2	M3	M4	M5
DATA	11.984	30.875	90.088	536.819	2356.922
ECOL	1.232	13.587	76.736	837.616	4566.336
HTOD	1.713	2.787	11.554	36.621	65.632
MATX	23.374	61.842	151.328	694.178	1757.023
NORB	0.007	0.010	0.024	0.065	0.132
PREC	3.207	0.292	24.351	73.607	169.439
SOLV	642.055	1969.218	5723.879	33861.239	71861.279
DTOH	0.004	0.006	0.021	0.072	0.098
Total	683.576	2078.671	6077.981	36040.217	80776.861

Table 6.15: Timing in seconds of each routine executed on a hybrid CPU/GPU platform with different nodal points for the 3D lid-driven cavity flow problem investigated at $Re = 1000$



	M1	M2	M3	M4	M5
CPU	1365.057	2850.348	5320.078	14173.739	32047.025
CPU/GPU	256.864	515.466	903.655	2368.717	5341.193
Speedup	5.314	5.530	5.887	5.984	6.000

Table 6.16: Comparisons of the total computation time(s) and the speedup for the 2D lid-driven cavity flow problem investigated at $Re = 1000$

	M1	M2	M3	M4	M5
CPU	4029.610	8250.572	14127.115	34027.644	66933.028
CPU/GPU	758.228	1423.955	2419.722	5569.455	10965.770
Speedup	5.315	5.794	5.838	6.110	6.104

Table 6.17: Comparisons of the total computation time(s) and the speedup for the 2D lid-driven cavity flow problem investigated at $Re = 5000$

	M1	M2	M3	M4	M5
CPU	2366.283	9605.931	29731.901	155611.971	580101.103
CPU/GPU	216.819	837.020	2498.637	11650.508	37474.785
Speedup	10.914	11.476	11.899	13.356	15.479

Table 6.18: Comparisons of the total computation time(s) and the speedup for the 3D lid-driven cavity flow problem investigated at $Re = 400$

	M1	M2	M3	M4	M5
CPU	8421.727	26894.400	79382.353	545168.143	1452458.737
CPU/GPU	683.576	2078.617	6077.981	36040.217	80776.861
Speedup	12.320	12.939	13.061	15.216	17.980

Table 6.19: Comparisons of the total computation time(s) and the speedup for the 3D lid-driven cavity flow problem investigated at $Re = 1000$

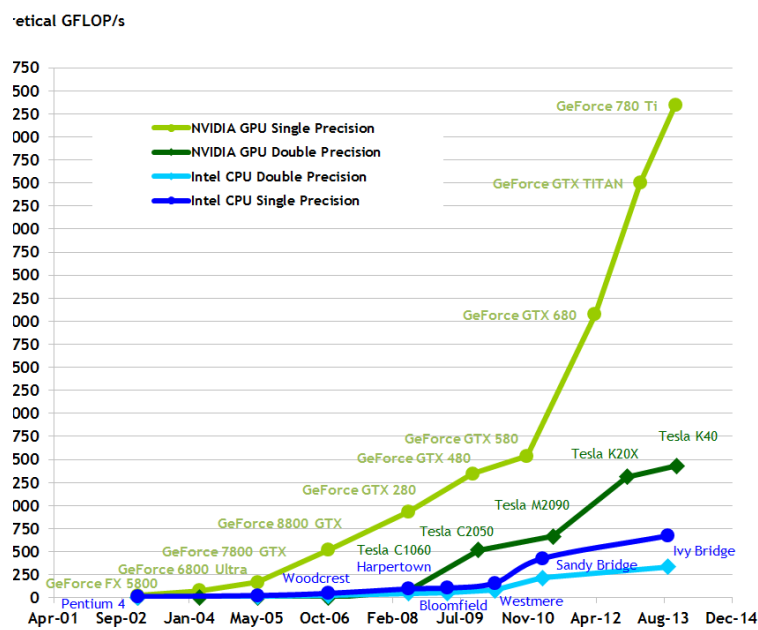


Figure 6.1: Comparison of the peak performance of the CPUs and the GPUs

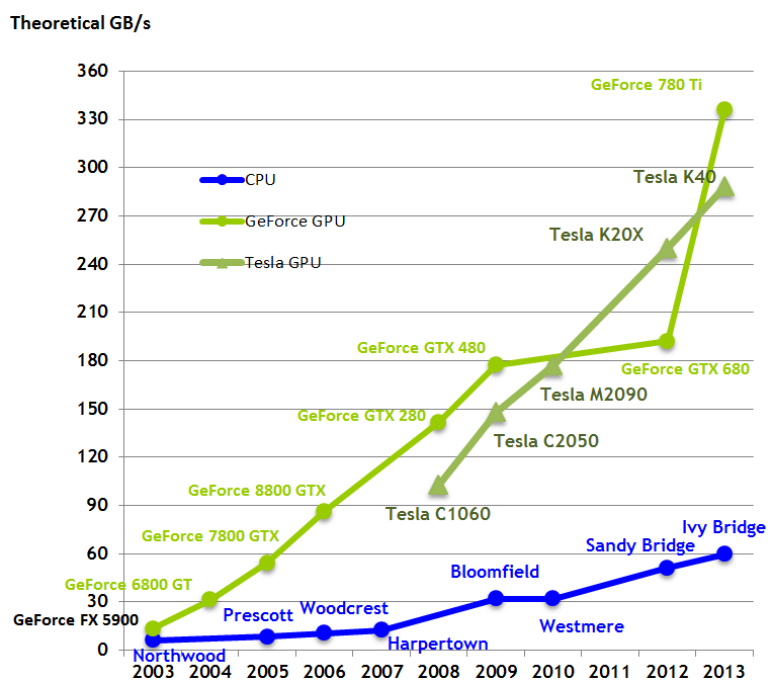


Figure 6.2: Comparison of the memory bandwidth of the CPUs and the GPUs

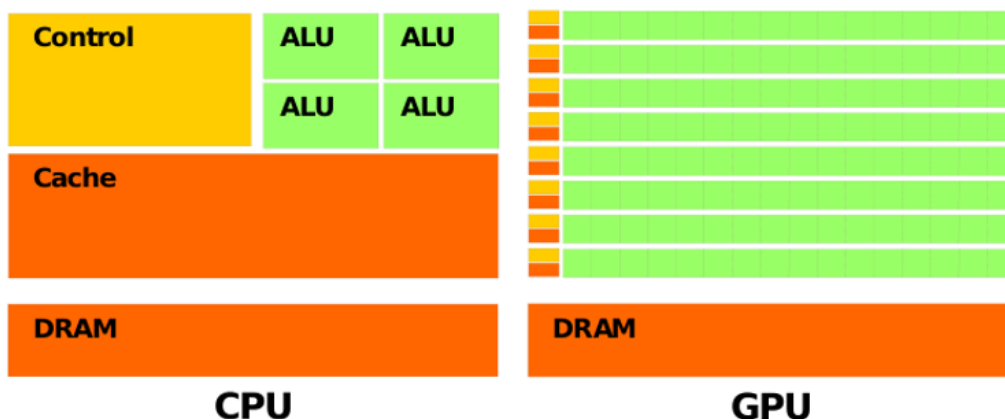
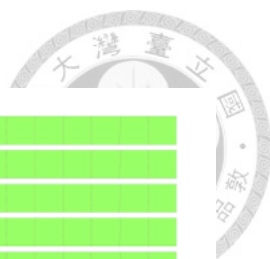


Figure 6.3: Basic structure of the two investigated typical processors [94]: (a) CPU ; (b) GPU

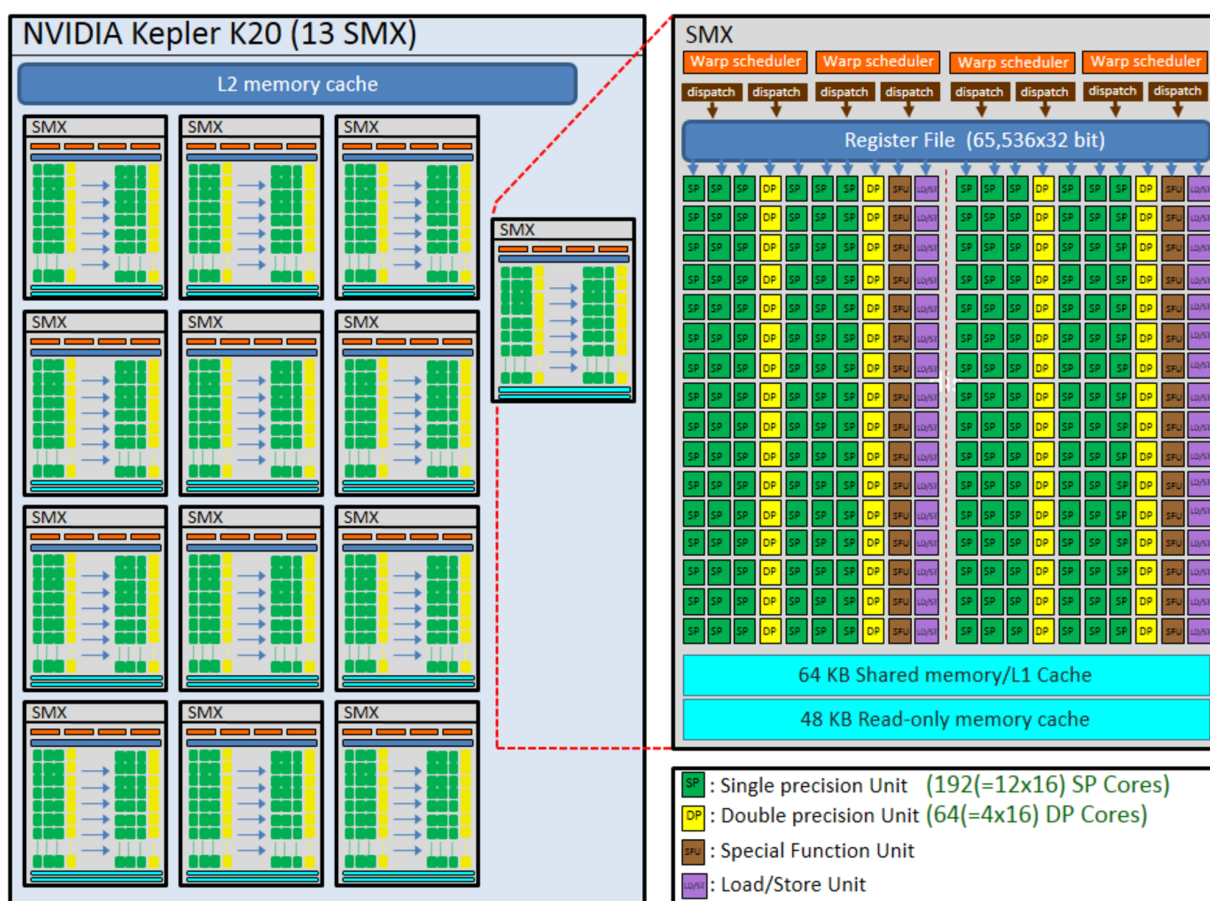


Figure 6.4: Basic structure of the Kepler K20 architecture.

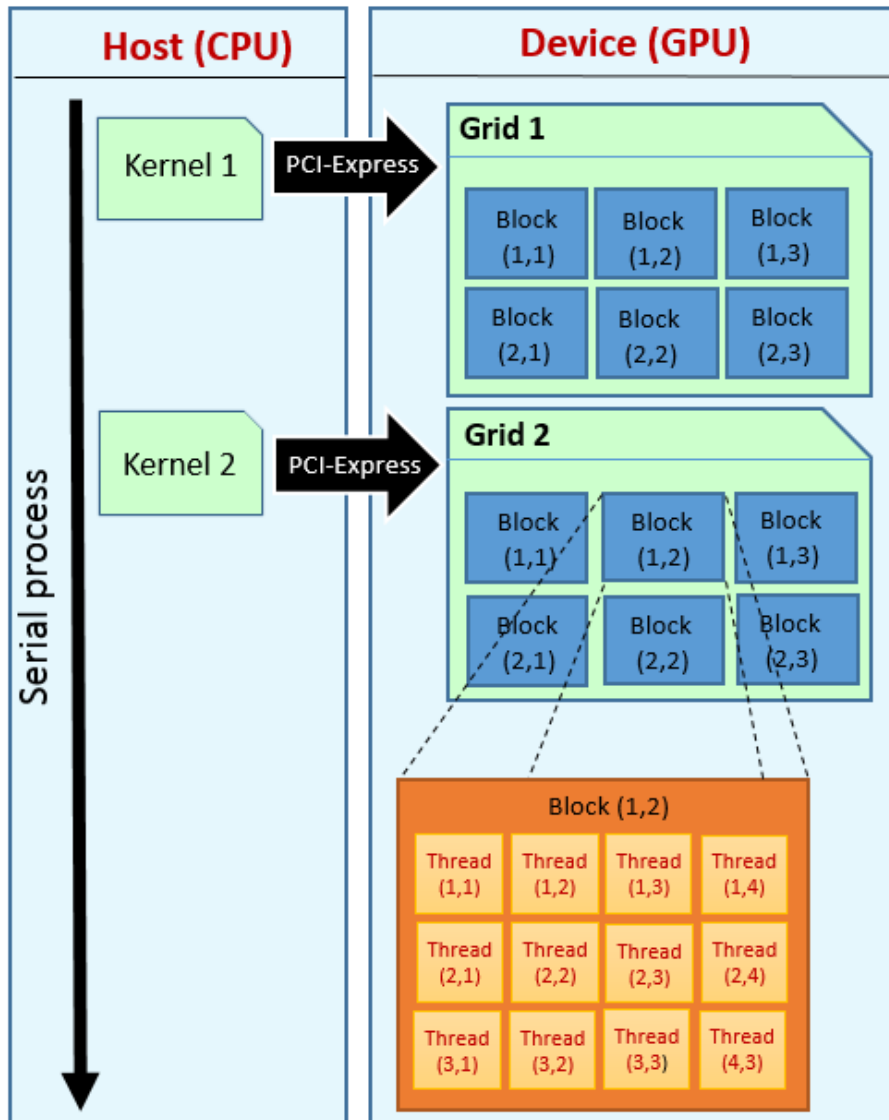


Figure 6.5: Schematic of the CUDA programming model.

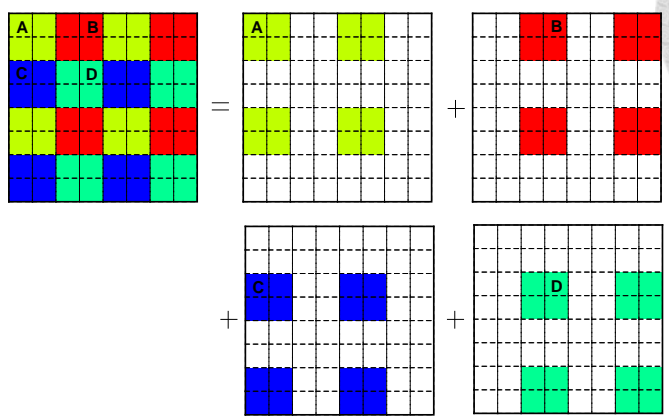
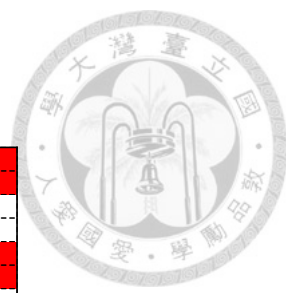


Figure 6.6: Illustration of the 2D mesh coloring strategy in four color-element groups without sharing the same global node.

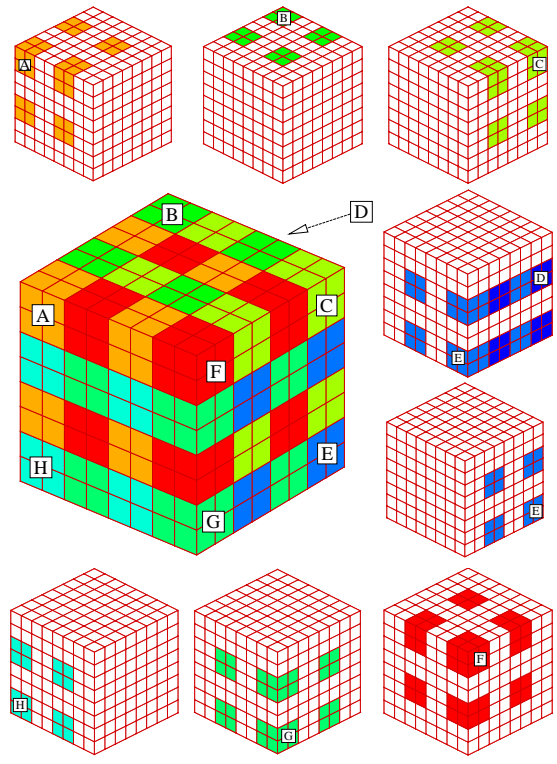


Figure 6.7: Illustration of the 3D mesh coloring strategy in eight color-element groups without sharing the same global node.

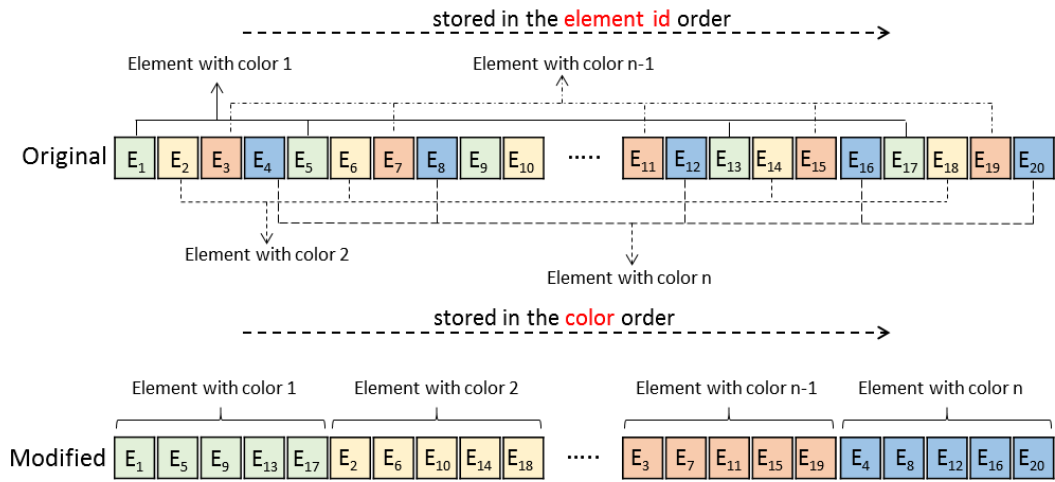


Figure 6.8: Distribution of all element matrices in global memory to satisfy the global memory coalescing.

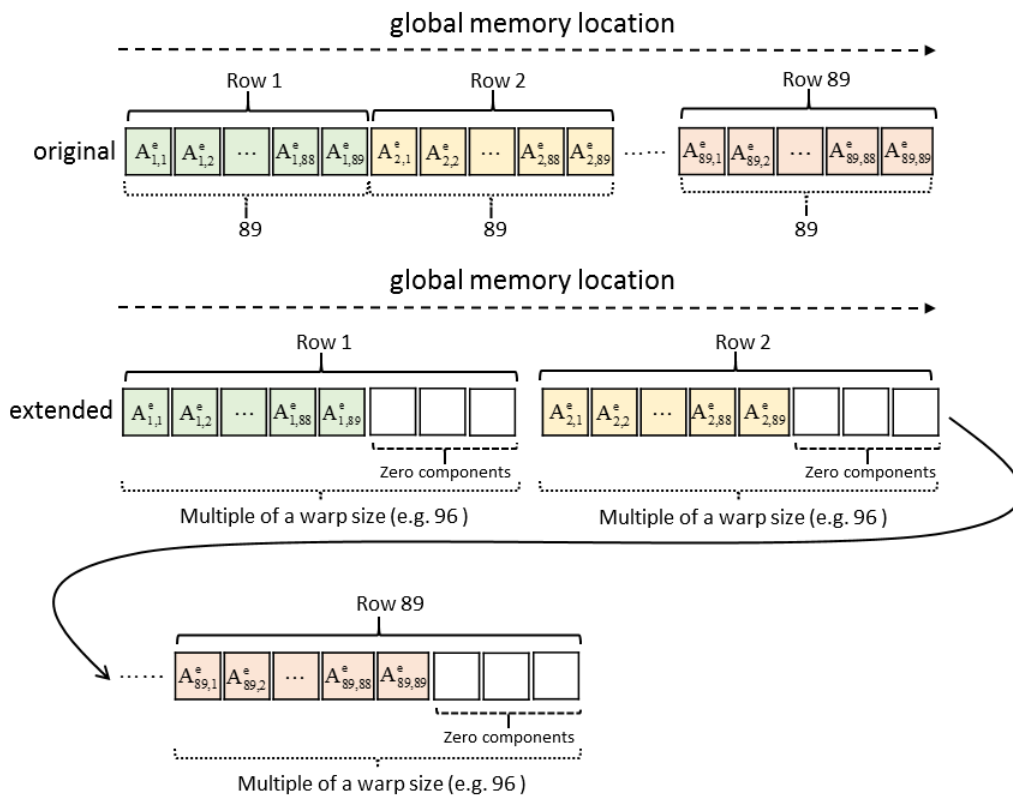


Figure 6.9: Extension of element matrix to satisfy the global memory coalescing condition.

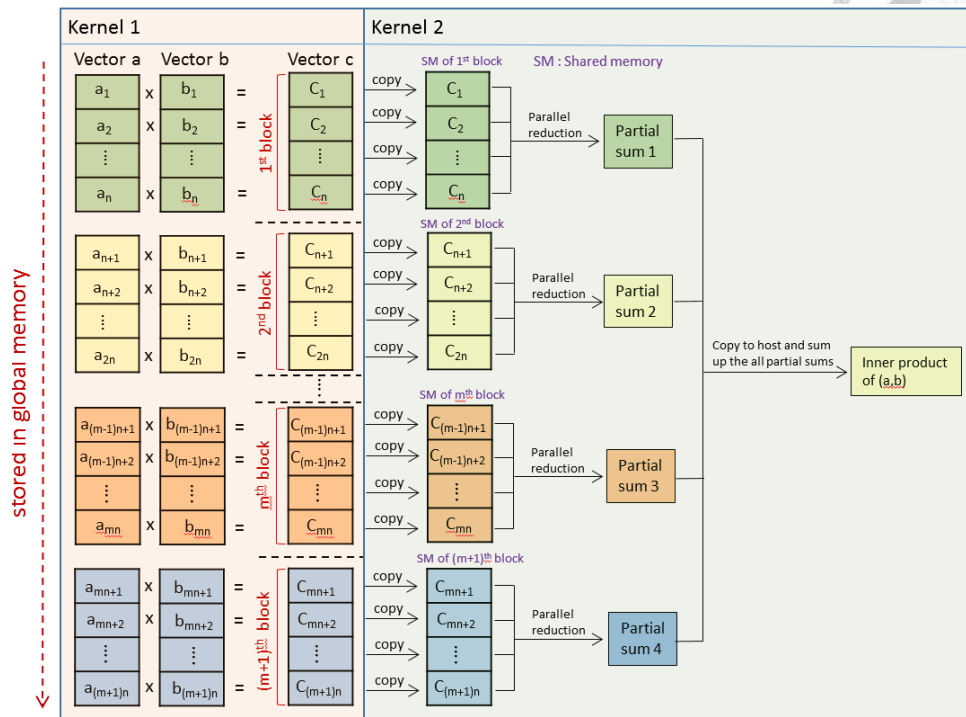


Figure 6.10: Illustration of the inner product operation on GPU

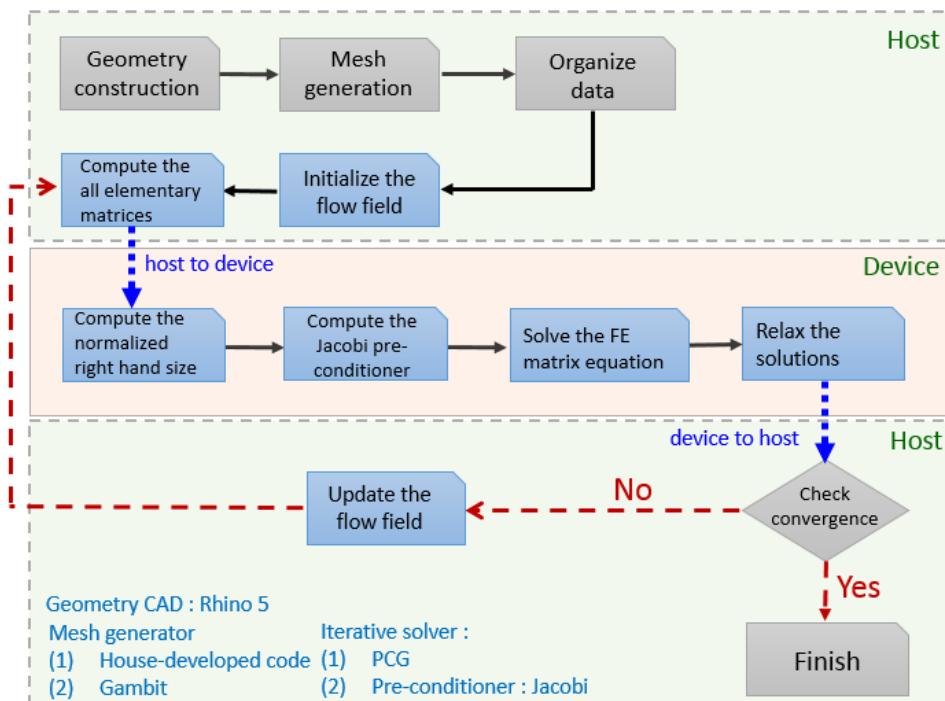
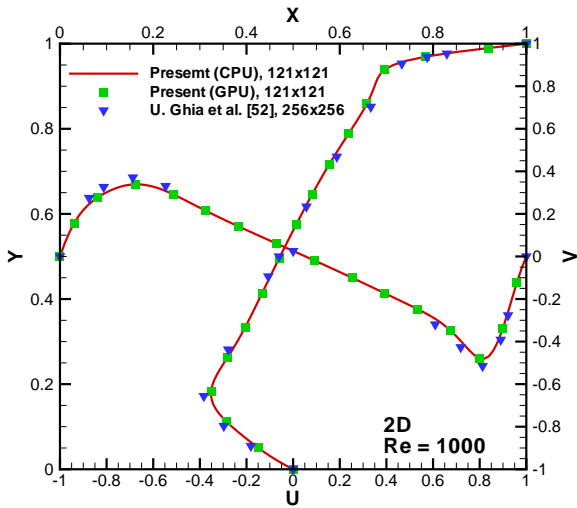
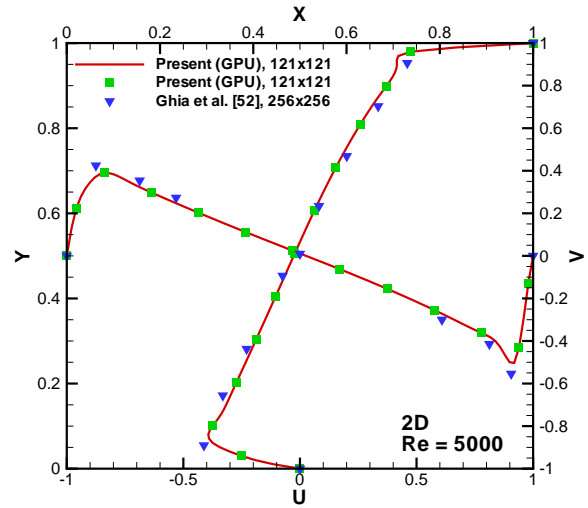


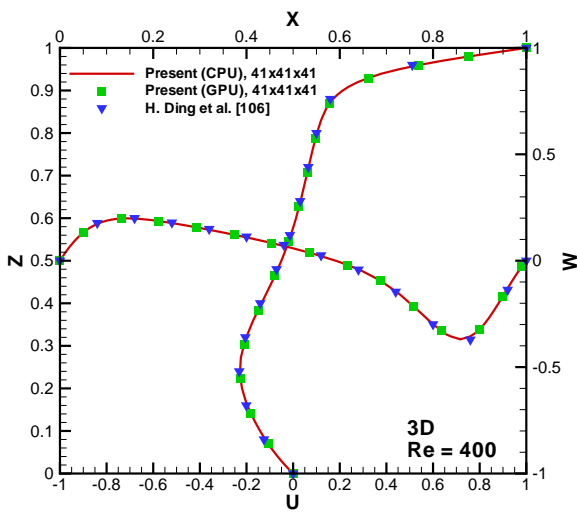
Figure 6.11: Flow chart for the developed code executed on a hybrid CPU/GPU platform.



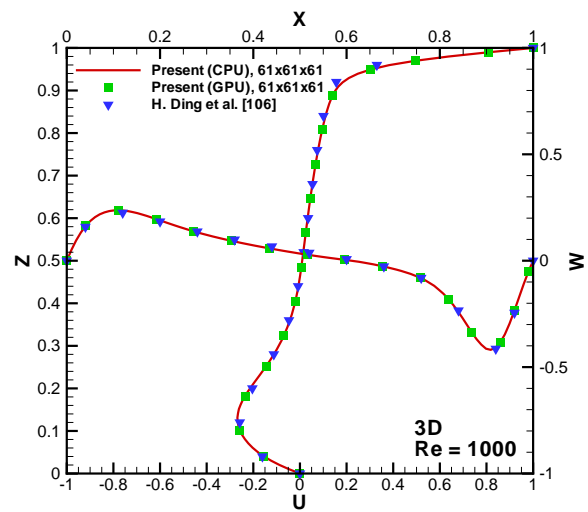
(a)



(b)

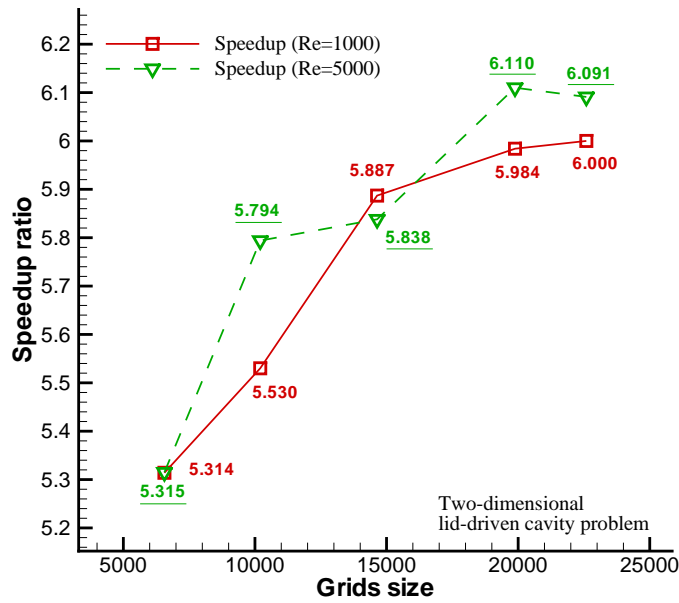


(c)

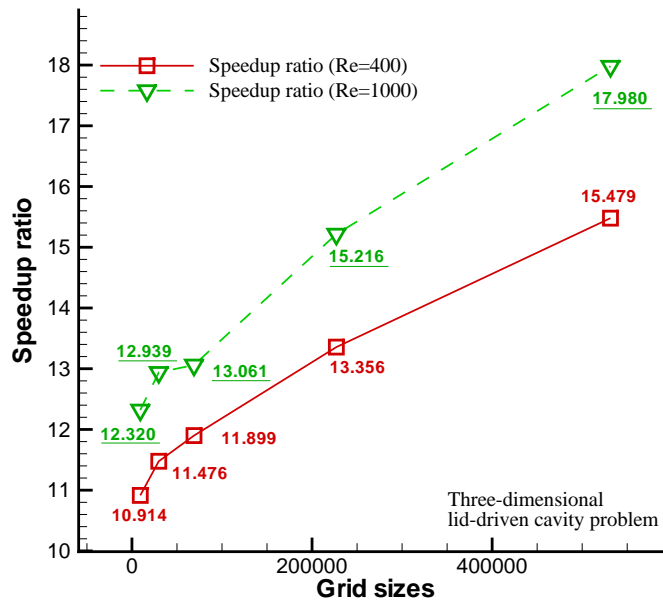


(d)

Figure 6.12: Comparison of the predicted mid-sectional velocity profiles for the lid-driven cavity problem considered in Sec. 6.8.3. (a)-(b) 2D problems ; (c)-(d) 3D problems.



(a)



(b)

Figure 6.13: Plot of speedup ratio for the two- and three-dimensional lid-driven cavity problem at different grids size.



Chapter 7

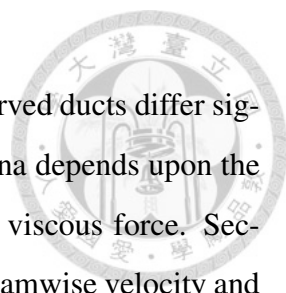
Applications

In previous chapters, we focused on developing a new finite element model for the simulation of incompressible fluid flow and heat transfer problems. The justification of using the proposed finite element model has been demonstrated through the several analytical verification and benchmark validation problems. In this chapter, the proposed finite element model will be applied to investigate some practical flow problems. In order to obtain the results quickly, all the calculations will be executed on CPU/GPU platform.

7.1 Three-dimensional 90 bend curved duct flow problem

Flow in curved ducts can be found in piping system, air conditioning, centrifugal pumps, aircraft intakes, river bends and cooling coils of heat exchanges. Their practical significance has motivated considerable research effort in the past. The destabilizing centrifugal and viscous (Tollmien-Schlichting) instabilities can be both present and they may furthermore interact with each other in the case of curved geometry. Due to nonlinear interaction between two instability mechanisms, it is highly possible that fluid flows in bent ducts might proceed prematurely from transition to turbulence. Advancing our understanding of the three-dimensional nature of the curved duct flow is, thus, of fundamental importance.

A main difference between curved and straight duct flows is the generation of secondary flow within the elbow region. It was firstly observed by Eustice in curved pipe [107,108]. Later on, the experimentally observed phenomena were analytically confirmed



by Dean [109]. The secondary flow motion that makes the flow in curved ducts differ significantly from that seen in straight ducts. Secondary flow phenomena depends upon the Dean number, which represents the ratio of centrifugal force to the viscous force. Secondary flow results in a pressure loss, the spatial redistribution of streamwise velocity and increased heat transfer with the duct. In addition, a much larger pressure drop, better mixing, and non-uniform wall shear stress are the major features of the fluid flow in curved ducts.

In the past decades, numerous studies of curved duct with rectangular or circle cross sections have been investigated. Due to the effects of geometry, the flow problems in rectangular curved ducts is more involved than those in the circle ducts. However, some useful insights are usually impossible to obtain in experiments. One way to obtain the secondary flow insight is to exploit the computational fluid dynamics technique. Study of this problem is particularly suitable to perform numerical simulation.

7.1.1 Problem description and validation

The curved duct with an unit rectangular cross-section schematically shown in Fig. 7.1 has a 90° bend of mean radius $R_c = 2.3$, is characterized by a dimensionless parameter $\bar{\delta} (\equiv r_o/r_i = 1.556)$. This geometric parameter plays an important role in affecting the balance of inertia, viscous, and centrifugal force. The elbow has upstream and downstream straight extensions. The downstream extension has a length 14 times the hydraulic diameter $D (= 1)$. It is, thus, rational to specify a fully-developed flow profile at the truncated outlet plane EFGH. Deflection of the prescribed inlet velocity may generate a strong transverse pressure gradient and, in turn, influence the streamwise pressure gradient. This can explain why the upstream extension is chosen to have a length of 2. With $U = 1$ as the characteristic velocity (inflow velocity), D a characteristic length and ν the fluid viscosity, the Reynolds number is obtained as $Re = 790$. The corresponding Dean number D_e , which is another dimensionless parameter used in the study of secondary flow, was 368.

The flow profile which is imposed at the inlet plane is taken as the fully-developed

laminar solution for a duct with a rectangular cross section [110].

$$u(\mathbf{y}, z) = \frac{48 \sum_{n=1,3,5,\dots} (-1)^{\frac{n-1}{2}} \left(1 - \frac{\cosh(n\pi\mathbf{y}/2h)}{\cosh(n\pi/2)}\right) \frac{\cos(n\pi z/2h)}{n^3}}{\pi^3 \left(1 - \frac{192}{\pi^5} \sum_{n=1,3,5,\dots} \frac{\tanh(n\pi/2)}{n^5}\right)} \quad (7.1)$$

where h is the half-height of duct. No-slip condition is imposed at all duct walls and the traction-free condition is prescribed at the outlet plane.

In view of the possibility of asymmetric solution, calculation is performed in the entire duct. The present calculations were performed in a non-uniform grid distribution. Comparison of the streamwise velocity profiles at different planes in Fig. 7.3 demonstrates that the calculation carried out in a grid involving 410375 nodal points is adopted according to the grid independence test. The residual reduction profile is plotted in Fig. 7.2.

For the validation purpose, we have compared the present finite element GPU solutions with the experimental data of Humphrey [111] and numerical data of Sotiropoulos [112]. Comparisons of the streamwise velocity profiles $u_\theta(\mathbf{r})$ within the elbow region was made at two planes $\mathbf{y} = 0.25$ and $\mathbf{y} = 0.5$, respectively. Good agreement is shown in Fig. 7.4 except at the plane $\theta = 60^\circ$. Like many numerical results have indicated, the reason for the discrepancy between the experimental and predicted results still remains unclear. Moreover, the velocity profiles $u_\theta(\mathbf{r})$ in the radial direction were also compared at different angle. As Fig. 7.5 shows, the agreement between the present numerical and experimental data is good except in the case of $\theta = 60^\circ$.

7.1.2 Numerical results

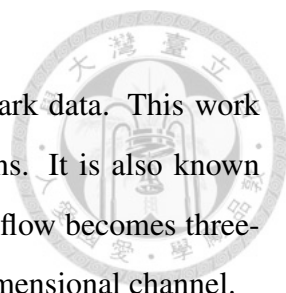
Discussion of the predicted numerical result begins with the axial flow development within the elbow region. In Fig. 7.4, the velocity profile in the vicinity of inner wall undergoes a rapid decrease, thus, forming a step-like profile. This results in a shift of the maximum axial velocity towards the outer wall due to the centrifugal force. Farther downstream, this step-like velocity profile continuously eroded to form a new local maximum and, thus, forming the M-shaped velocity profile with two local maximums. This

flow phenomenon was experimentally confirmed by Humphrey. In addition, the step-like velocity profile near the inner wall was also experimentally observed by Agrawal [113] and numerically simulated by Soh [114].

Fig. 7.6 shows the pressure contours $p(r)$ at three different planes. In Fig. 7.6, the difference between the high pressure near outer wall and low pressure near the inner wall leads to an inwardly directed pressure gradient. As the flow enters the curved duct, an acceleration flow develops immediately in regions near the inner wall due to the inwardly directed pressure gradient. Conversely, deceleration flow will occur in regions near the outer wall because of the adverse pressure gradient established in the downstream region after about $\theta = 0^\circ$. The imbalance of inwardly pressure gradient and centrifugal force drives fluid flow in the elbow region to move toward to the outer wall and return to the inner wall along the duct wall, resulting in a vortex pair. One vortex is symmetric to the other with respect to the plane of symmetry. The axial flow is superimposed over the secondary flow, thereby forming a flow that is of substantially three-dimensional nature in the elbow region, as schematically shown in Fig. 7.7. Also, the predicted velocity vectors are plotted at different chosen planes in Fig. 7.8 for illustrating the formation of the secondary flow patterns.

7.2 Three-dimensional backward facing step flow problem

Fluid flow in the channel with flow separation, flow reattachment and multiple recirculation bubble are commonly encountered in many real flow problems. Some examples are the flow over airfoils at large angle of attack, channel flow with area being suddenly increased, and the flow in heat transfer devices. Among this type of flow problem, the backward-facing step flow problem was regarded as having the simplest geometry while retaining many rich and interesting flow phenomena. This flow problem has been extensively studied, for instance, under different Reynolds numbers [115], step height and expansion ratio [116], inlet entrance effect [62, 117] and inflow/outflow boundary condition [57, 58, 118]. Armaly et al. [119] have conducted the experiments on the backward-



facing step flow and have presented valuable experimental benchmark data. This work was extensively employed to validate and verify numerical solutions. It is also known from this study that as the Reynolds number is greater than 400 the flow becomes three-dimensional, yielding different results than those obtained in two-dimensional channel.

7.2.1 Problem description and numerical results

In this study, problem geometry and flow conditions reported by Armaly are considered due to the available experiment data for making a direct comparison of results. In Fig. 7.9, the channel upstream height, h , is 1 and the channel downstream height H is 1.9423, giving an expansion ratio $H/h = 1.9423$ and the step has a height $S = 0.9423$. The channel width is $35h$ and the downstream length $L_d = 50h$. The fully-developed analytical velocity profile with the mean velocity $u_{mean} = 1$ is imposed at the inlet boundary. No-slip condition is imposed at the channel walls and traction-free condition is imposed at the channel outlet plane. In this study, the full-scaled calculation is performed rather than in half channel with the symmetric mid-plane. The calculations at $Re = 100, 389, \text{ and } 1000$ are carried out in a grid involving 496131 nodal points. The residual reduction profiles for the cases $Re=100$ and 389 are plotted in Fig. 7.10.

For the sake of validation purpose, the streamwise velocity profiles for $Re = 100, 389$ and 1000 at mid-plane are plotted in Fig. 7.11 Included in the same figure are the experimental data of Armaly [119] and the two-dimensional numerical results for comparative purpose.

In case the Reynolds number is less than 400, there is an excellent agreement between the two- and three-dimensional predicted solutions and the experimental data. These results reveal that the channel width $35h$ is sufficient wide to ignore the left and right-wall effect in the immediate vicinity of mid-plane.

When the Reynolds number is larger than 400, only the predicted two-dimensional solutions is particular apparent in the range of $10h \leq x \leq 35h$. This result was also pointed out by Chiang et al. and they stated that the main reason of this discrepancy is due to the effect of roof eddy in the three-dimensional analysis

Moreover, the predicted reattachment lengths x_1 at the mid-plane is compared with the experimental data of Armaly [119]. The good agreement in the Reynolds number range of $100 \leq Re \leq 800$ in Fig. 7.12.



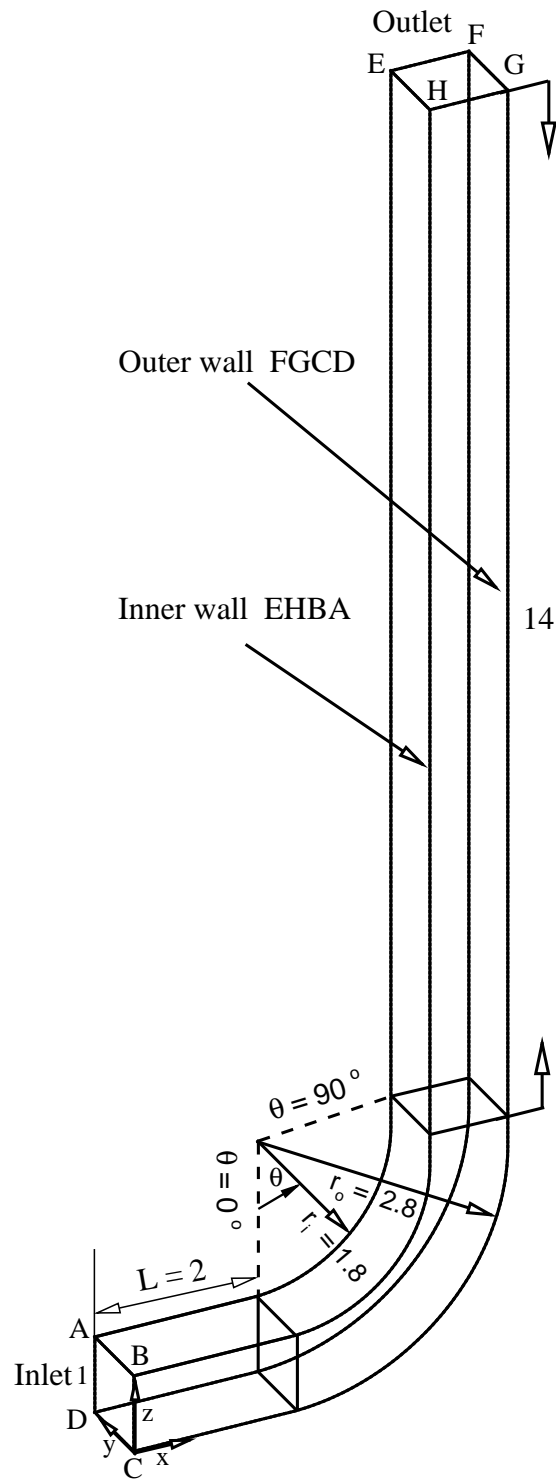


Figure 7.1: Schematic of the 90 bend duct flow problem considered in Sec. 7.1

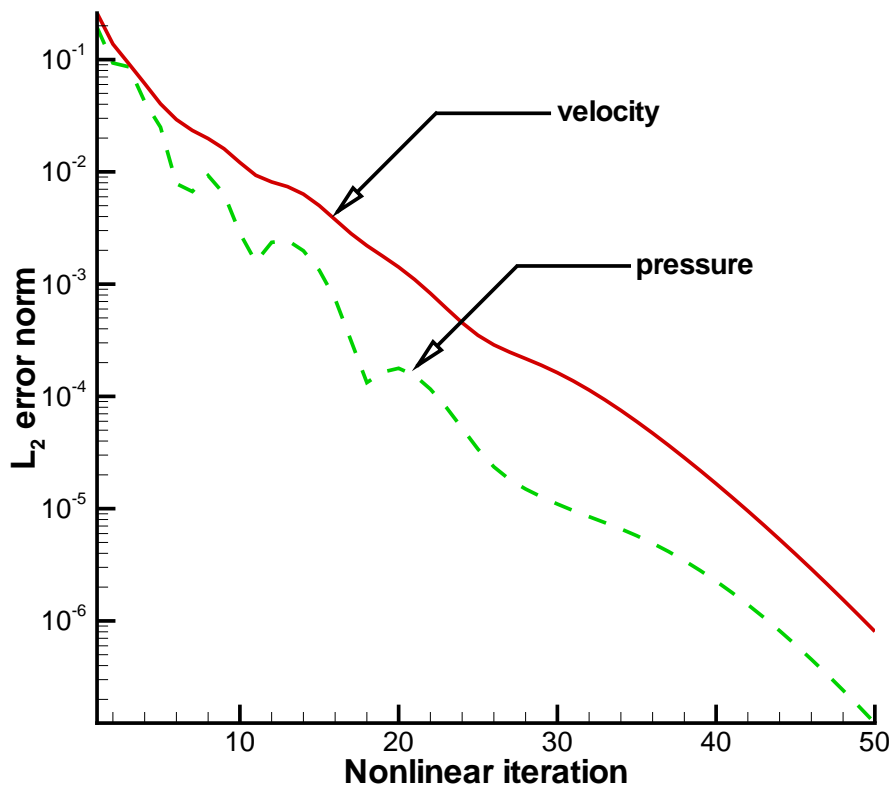


Figure 7.2: Residual reduction plot for the 90-degree bend duct flow problem considered in Sec. 7.1. (grids : 410375)

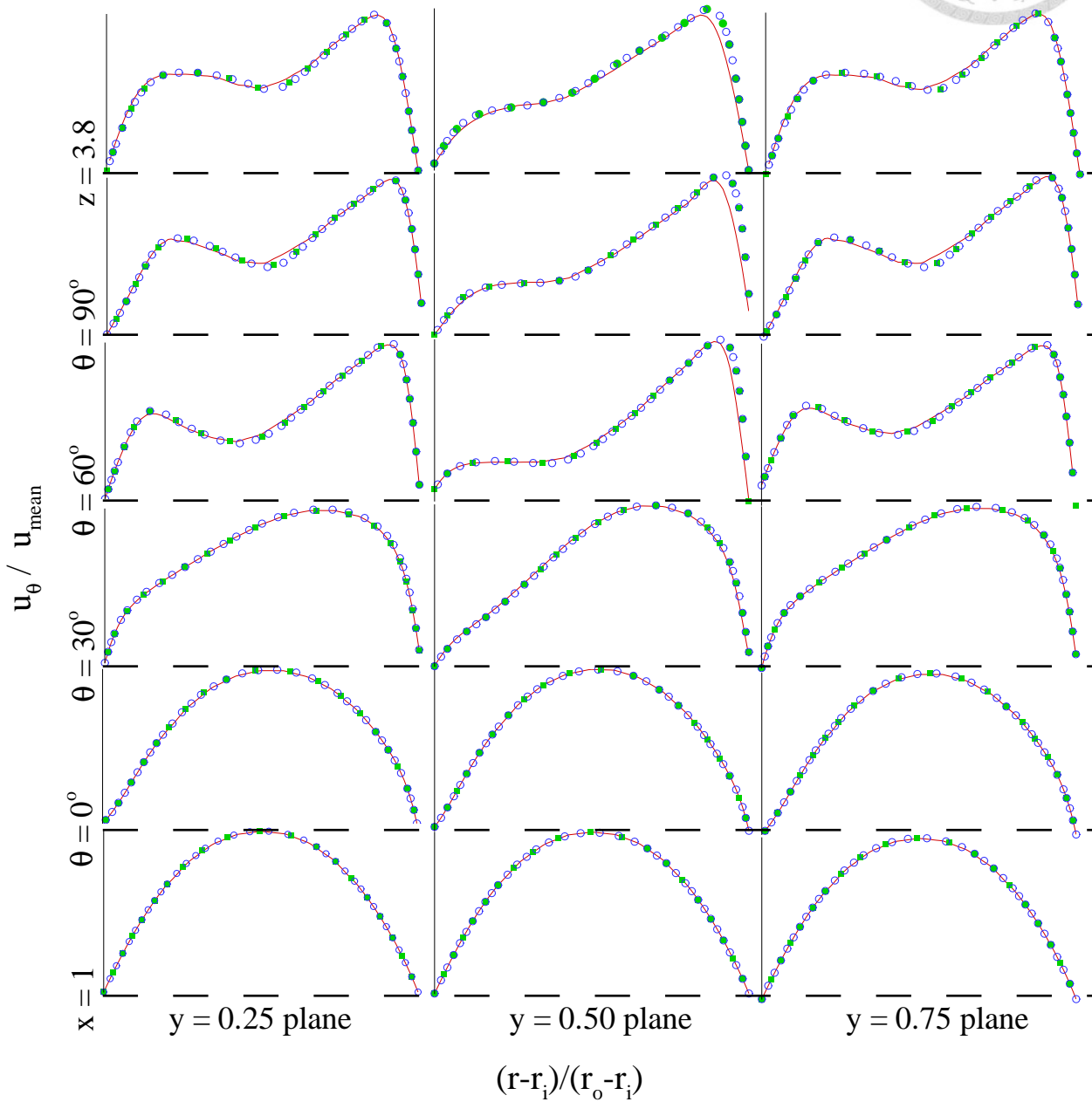


Figure 7.3: Grid independence test for the 90-degree bend duct flow problem considered in Sec. 7.1. — grids 327789 ; ■ grids 410375 ; ○ grids 464275

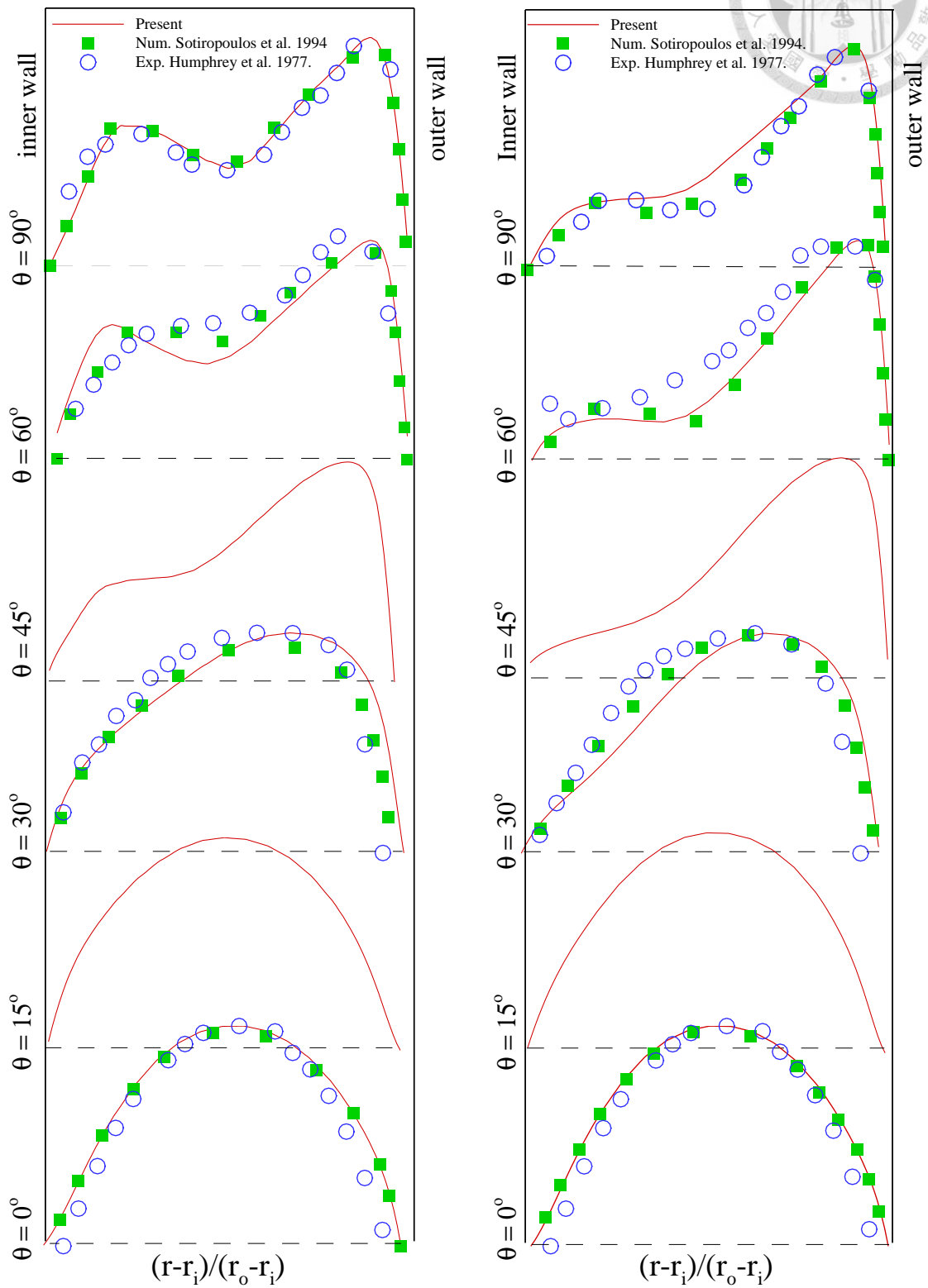


Figure 7.4: Comparison of the predicted velocity profiles $u_\theta(r, y)$ with other two solutions obtained at different angles θ defined in Fig. 7.1 (a) $y = 0.25$ plane ; (b) $y = 0.5$ plane.

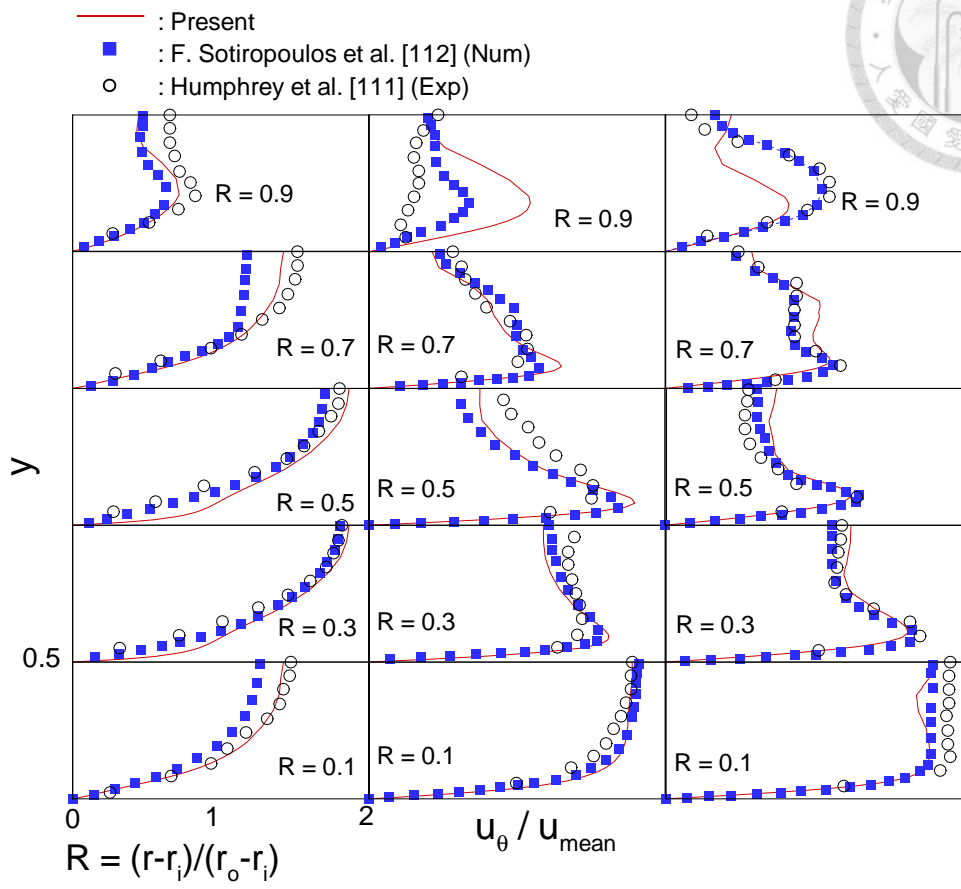


Figure 7.5: Comparison of the predicted axial velocity profiles $u_{\theta}(r,y)$ with other two solutions obtained at different angles θ defined in Fig. 7.1

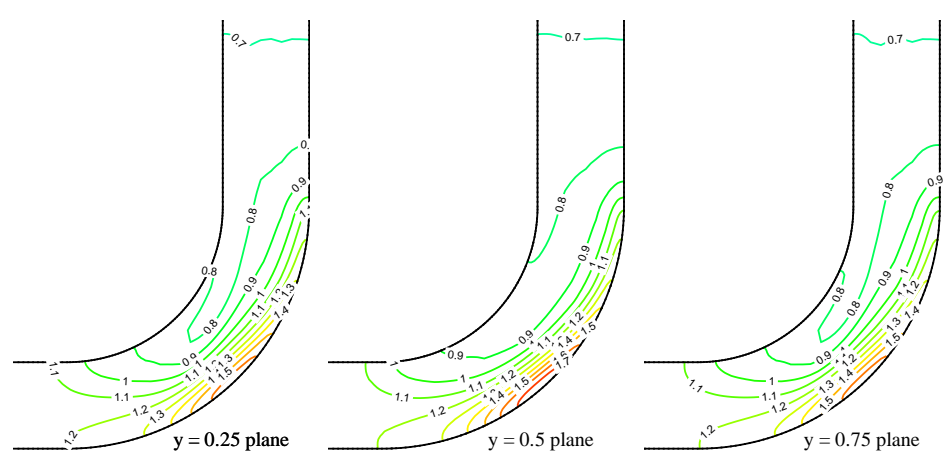
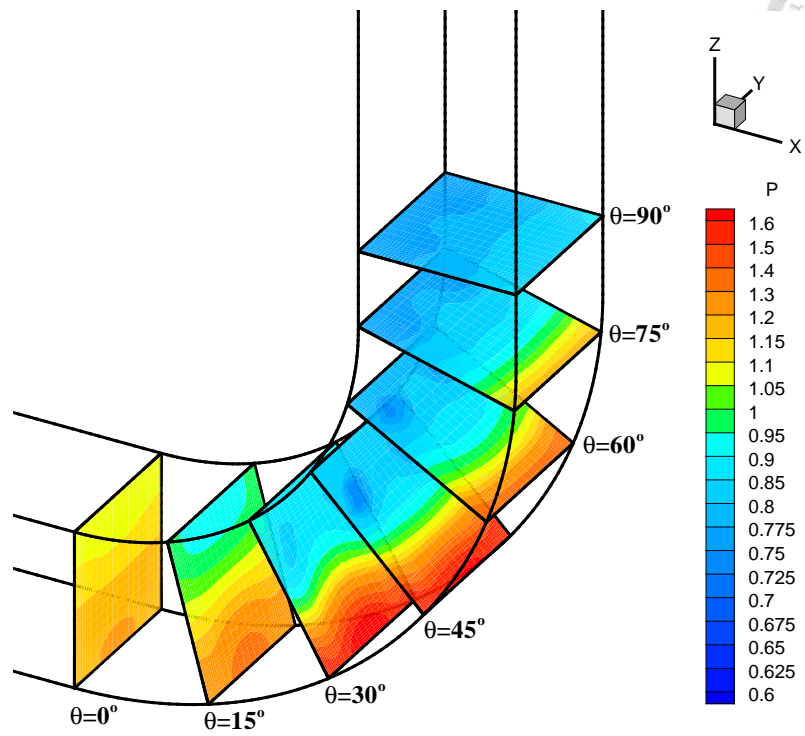
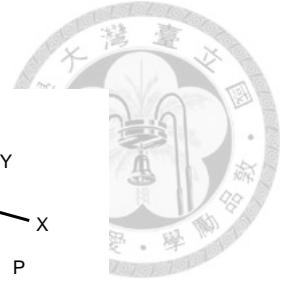
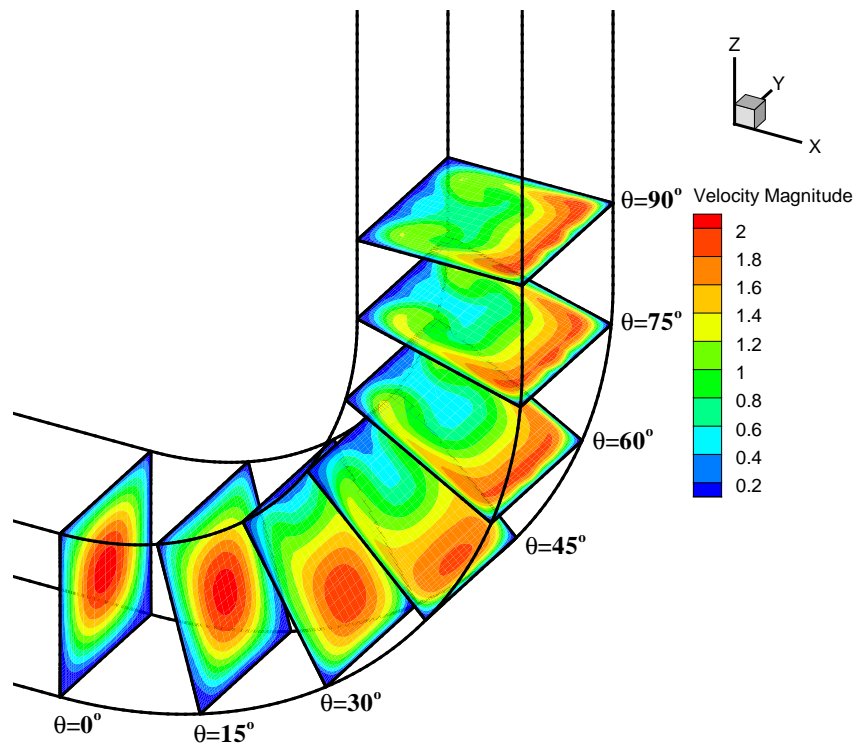


Figure 7.6: Pressure contours plots at three different planes

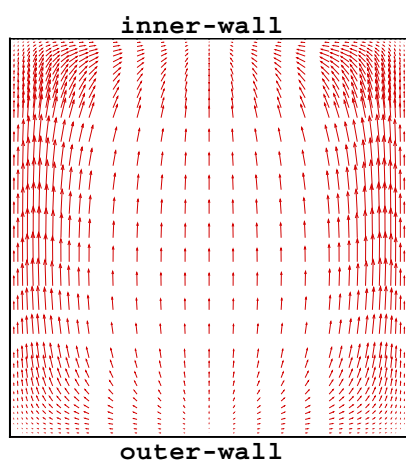
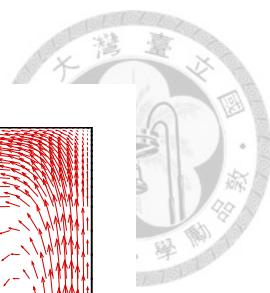


(a)

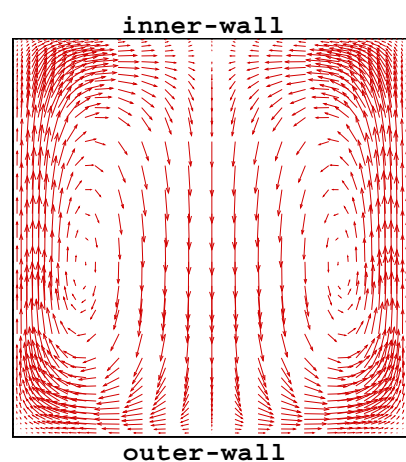


(b)

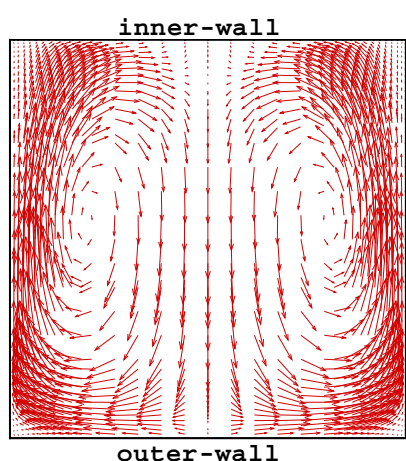
Figure 7.7: Velocity and pressure contours within the elbow region.



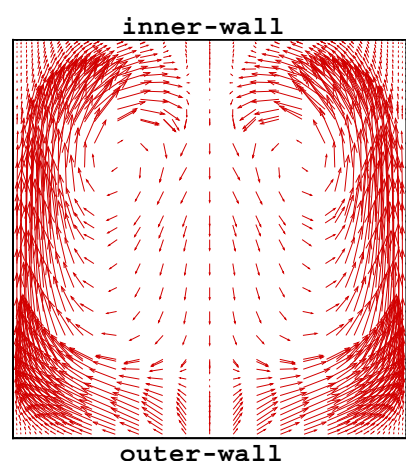
(a)



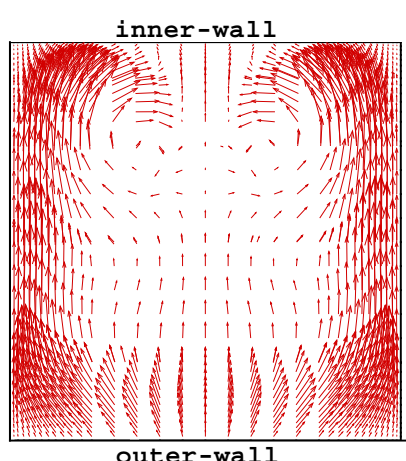
(b)



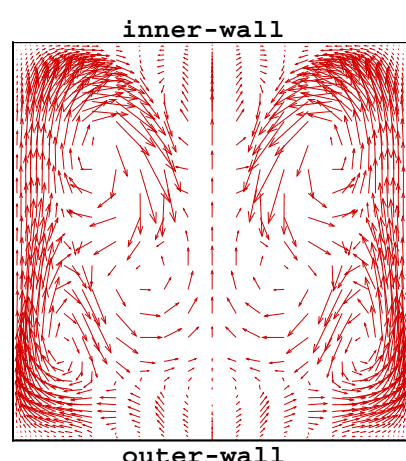
(c)



(d)



(e)



(f)

Figure 7.8: Velocity vector plots at different planes for the 90-degree bend duct flow problem considered in Sec. 7.1

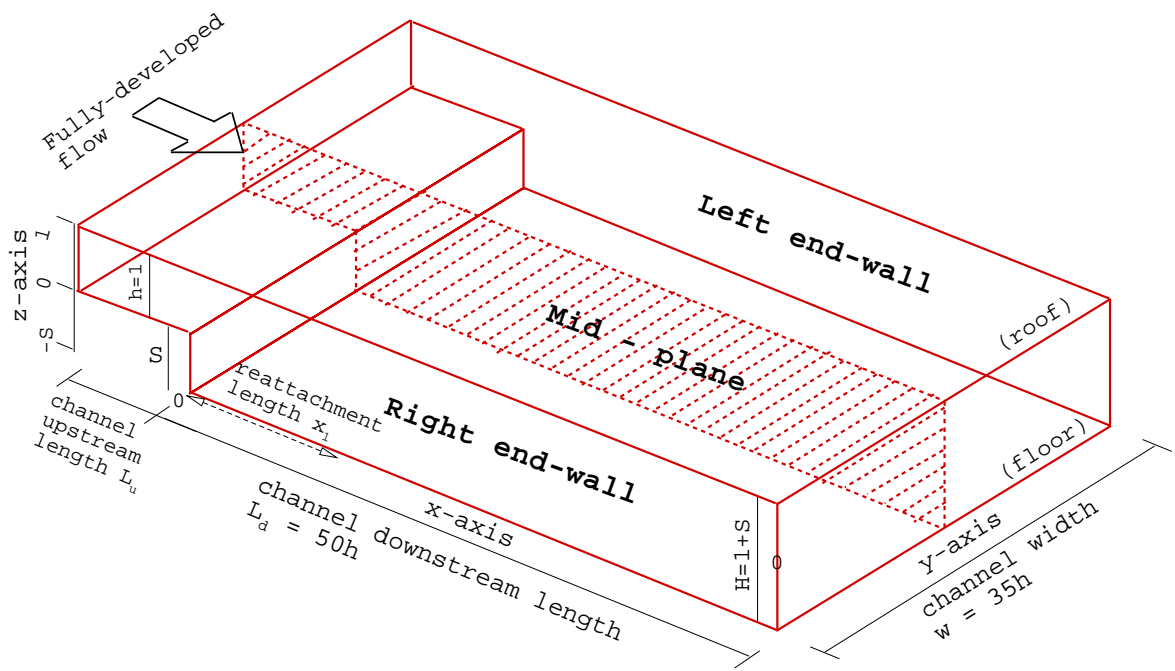
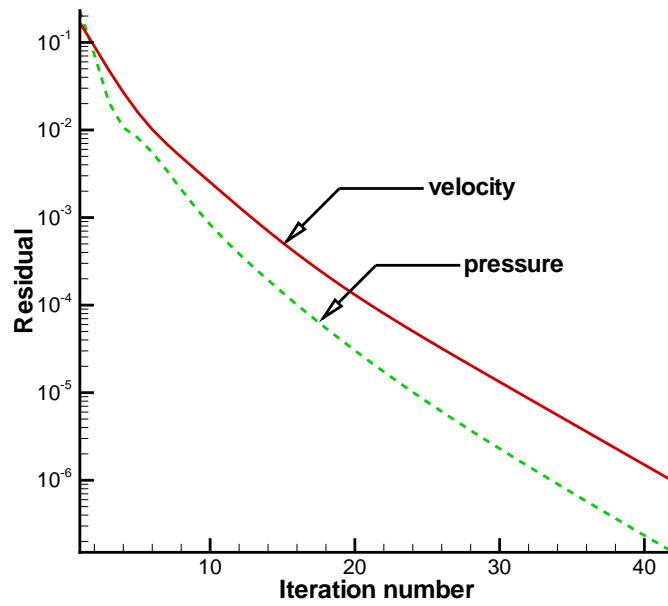
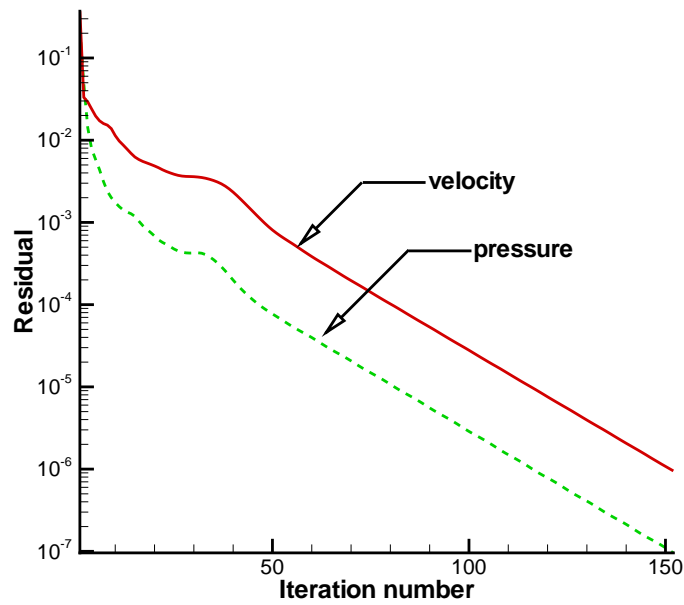


Figure 7.9: Schematic of the three-dimensional backward-facing step flow problem considered in Sec. 7.2

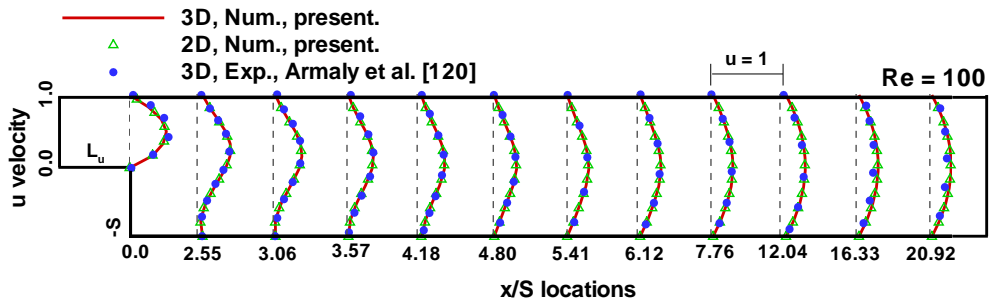


(a)

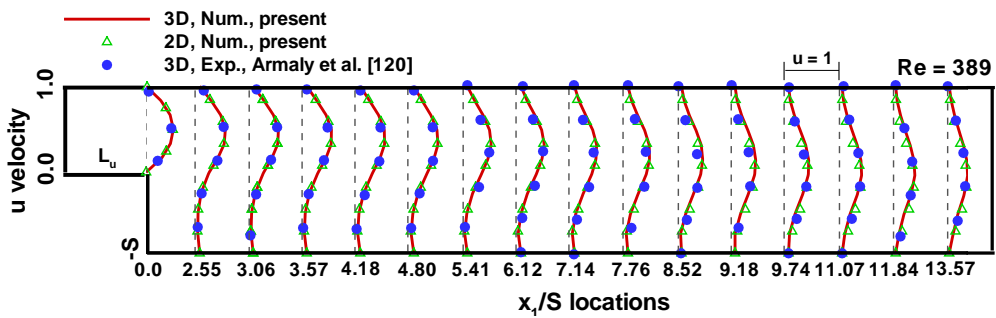


(b)

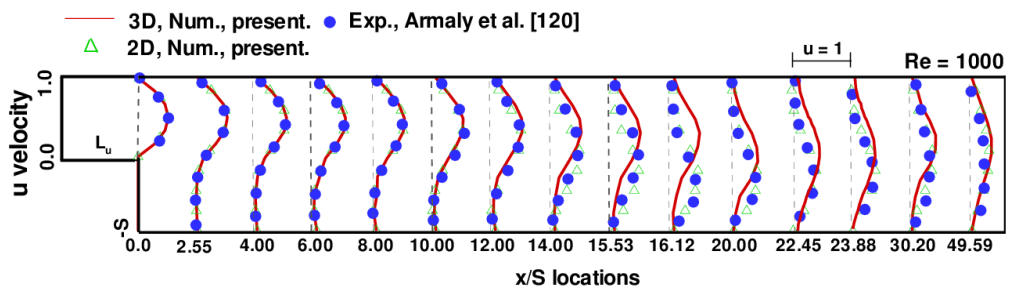
Figure 7.10: Residual reduction plots for the three-dimensional backward-facing step flow problem considered in Sec. 7.2. (a) $Re = 100$; (b) $Re = 389$.



(a)



(b)



(c)

Figure 7.11: Comparisons of the streamwise velocity profiles for the three-dimensional backward-facing step flow problem considered in Sec. 7.2. (a) $Re = 100$; (b) $Re = 389$; (c) $Re = 1,000$.

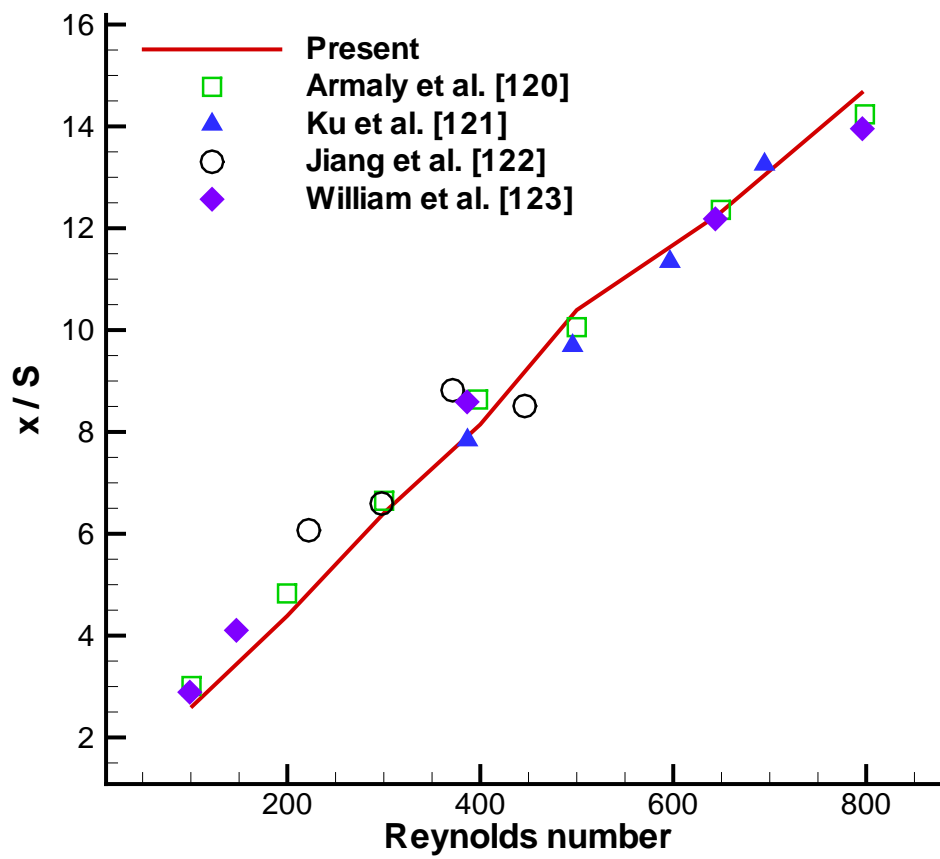


Figure 7.12: Comparison of the reattachment lengths x_1/h with the results of Armaly et al. [119] for the three-dimensional backward-facing step flow considered in Sec. 7.2



Chapter 8

Concluding Remarks

In this study, we have focused on developing a new GPU-based parallel finite element model to predict two- and three-dimensional incompressible viscous flow and heat transfer problems. The developed model is endowed with the minimized wavenumber error property and an oscillations suppressant capability to suppress erroneous oscillations arising from the non-symmetric convection term at high Reynolds numbers. The \mathcal{LBB} satisfied interpolation functions are chosen to prevent oscillation of the computed pressure fields. Good agreements between the numerical results and the solution to the benchmark problems show that the proposed finite element model is stable and reliable.

In the three-dimensional mixed finite element formulation, the use of an iterative solver may fail for solving the unsymmetric and indefinite finite element equations. In order to enhance the stability, we advocated the use of normalization strategy and the preconditioned conjugate gradient solver to increase stability during convergence. Two polynomial-based pre-conditioners for the normalized system are presented to reduce the increasing condition number due to the normalization procedure. To avoid assembling the global matrix, the use of EBE technique and the implementation of boundary condition on element-level matrices are also presented. The efficiency of the proposed strategy is confirmed via several numerical experiments.

With the advent of CUDA, the utilization of GPU devices for performing general scientific and engineering computation becomes a possibility. With recent GPU hardware, such as NVIDIA's Kepler architecture used in this thesis, the finite element parallel calculations were executed. Some optimization strategies are introduced in order to maximize

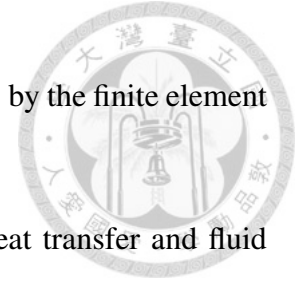
the GPU's computational power. We believe that the true potential of GPU can be effectively tapped as the problem sizes increase. The future work of this topic is to extend the computation from single CPU to multiple CPUs. This can be done via the domain decomposition method and MPI framework.





Bibliography

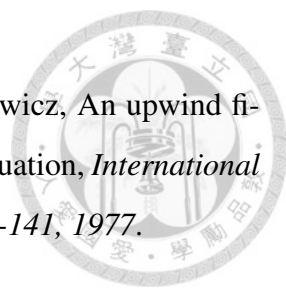
- [1] R. Aris, Vector, Tensor, and the basic equations of fluid mechanics, *Prentice-Hall Cliffs, Nj, 1962.*
- [2] G. K. Batchelor, An introduction to fluid dynamics, *Cambridge University Press, 1967.*
- [3] A. J. Chorin, J. E. Marsden, A mathematical introduction to fluid mechanics, *Springer-Verlag press, New York, 1998.*
- [4] C. A. J. Fletcher, Computational Techniques for Fluid Dynamics Volume I, *Springer-Verlag press, New York, 1990.*
- [5] R. H. Pletcher, J. C. Tannehill, D. Anderson, Computational fluid mechanics and Heat transfer, *CRC press, 2012.*
- [6] R. Curren, Variational methods for the solution of problems of equilibrium and vibration, *Bulletin American Mathematical Society, pp.1-23, 1943.*
- [7] S. C. Brenner, L. R. Seott, The mathematical theory of finite element method, *Springer-Verlag press, New York, 1994.*
- [8] P. G. Ciarlet, The finite element method for elliptic problem, *Holland North Amsterdam, 1978.*
- [9] G. S. Almasi, Highly Parallel Computing, *Benjamin-Cummings publisher, Redwood city publishers, 1998.*

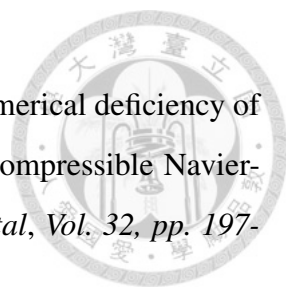


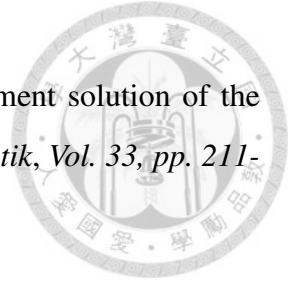
- [10] C. Johnson, Numerical solution of partial differential equations by the finite element method, *Cambridge University Press*, 1992.
- [11] J. N. Reddy, D. K. Gartling, The finite element method in heat transfer and fluid dynamics, *CRC press*, 2010.
- [12] A. J. Baker, Finite element computational fluid mechanics, *McGraw-Hill, New York*, 1984.
- [13] L. P. Franca, G. Hauke, A. Masud, Revisiting stabilized finite elements for the advective-diffusive equation, *Computer Methods in Applied Mechanics and Engineering*, Vol. 195, pp. 1560-1572, 2006.
- [14] J. Donea, A. Huerta, Finite element methods for flow problems, *John Wiley press*, 2003.
- [15] M. J. Quinn, Parallel programming in C with MPI and OpenMP, *McGraw-Hill, New York*, 2004.
- [16] NVIDIA, CUDA (Compute Unified Device Architecture) programming guide, *NVIDIA*, 2007.
- [17] H. Elman, D. Silvester, A. Wathen, Finite elements and fast iterative solver with applications in incompressible fluid dynamics, *Oxford University Press, United Kingdom*, 2014.
- [18] L. Navier, Mémoire sur les lois du mouvement des fluides, *Mén. del'Acad. des Sci.* pp. 389, vi(1822).
- [19] G. G. Stokes, On the Theories of the Internal Friction of Fluids in Motion, and of the Equilibrium and Motion of Elastic Solids, *Mathematical and Physical Papers, Vol. 1*, pp. 75-129, 2009.
- [20] L. Quartapelle, Numerical solution of the incompressible Navier-Stokes equations, *Birkhäuser-Verlag, Basel, Boston, Berline*, 1993.



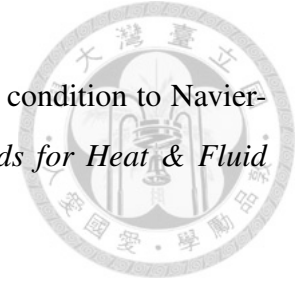
- [21] S. V. Patankar, D. B. Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flow, *International Journal Heat Mass Transfer*, Vol. 15, pp.1787-1806, 1972.
- [22] S. V. Patankar, Numerical heat transfer and fluid flow, *New York, McGraw-Hill*, 1980.
- [23] S. V. Patankar, A calculation procedure for two-dimensional elliptic simulation, *Numerical Heat Transfer*, Vol. 4, pp. 409-425, 1981.
- [24] J. P. Von Doormaal, G. D. Raithby, Enhancement of the SIMPLE method for predicting incompressible fluid flow, *Numerical Heat Transfer*, Vol. 7, pp. 147-163, 1984.
- [25] A. J. Chorin, Numerical solution of the Navier-Stokes equations, *Mathematics of Computation*, Vol. 22, pp. 745-762, 1968.
- [26] R. Teman, Une méthode d'approximation de la solution des équations de Navier-Stokes, *Bulletin de la Societe Mathematique de France*, Vol. 96, pp. 115-152, 1968.
- [27] O. C. Zienkiewicz, P. Nithiarasu, R. Codina, Vázquez, The characteristic based split procedure: An efficient and accurate algorithm for fluid problems, *International Journal for Numerical Methods in Fluids*, Vol. 31, pp. 359-392, 1999.
- [28] N. Massarotti, F. Arpino, R. W. Lewis, P. Nithiarasu, Explicit and semi-implicit CBS procedures for incompressible viscous flows, *International Journal for Numerical Methods in Engineering*, Vol. 66, pp. 1618-1640, 2006.
- [29] K. W. Morton, Numerical solution of convection-diffusion problems, *Chapman & Hall, London*, 1996.
- [30] I. Christie, D. G. Griffiths, A. R. Mitchell, O. C. Zienkiewicz, Finite element methods for second order differential equation with significant first-order derivative, *International Journal for Numerical Methods in Engineering*, Vol. 10, pp. 1389-1396, 1976.

- 
- [31] J. C. Heinrich, P. S. Huyaakorn, A. T. Mitchell, O. C. Zienkiewicz, An upwind finite element scheme for two-dimensional convective transport equation, *International Journal for Numerical Methods in Engineering*, Vol. 11, pp. 131-141, 1977.
- [32] J. C. Heinrich, O. C. Zienkiewicz, Quadratic finite element scheme for two-dimensional convection transport problems, *International Journal for Numerical Methods in Engineering*, Vol. 11, pp. 1831-1844, 1977.
- [33] T. J. R. Hughes, A simple scheme for development 'upwind' finite element, *International Journal for Numerical Methods in Engineering*, Vol. 12, pp. 1359-1365, 1978.
- [34] A. N. Brook, T.J.R. Hughes, Streamline upwind Petrov/Galerkin formulation for convection dominated flow with particular emphasis on the incompressible Navier-Stokes equations, *Computer Methods in Applied Mechanics and Engineering*, Vol. 32, pp. 199-259, 1982.
- [35] A. Mizukami, T.J.R. Hughes, A Petrov-Galerkin finite element method for convection dominated flow: An accurate upwinding technique for satisfying the maximum principle, *Computer Methods in Applied Mechanics and Engineering*, Vol. 50, pp. 181-193, 1985.
- [36] T. J. R. Hughes, M. Mallet, A. Mizukami, A new finite element formulation for computational fluid dynamics : II. Beyond SUPG, *Computer Methods in Applied Mechanics and Engineering*, Vol. 54, pp. 341-355, 1982.
- [37] S. F. Tsai, Tony W. H. Sheu, Finite-element analysis of incompressible Navier-Stokes equations involving exit pressure boundary conditions, *Numerical Heat Transfer, Part B : Fundamental*, Vol. 39, pp. 479-507, 2001.
- [38] C. K. W. Tam, J. C. Webb, Dispersion-relation-preserving finite difference scheme for computational acoustics, *Journal of Computational Physics*, Vol. 107, pp. 262-281, 1993.

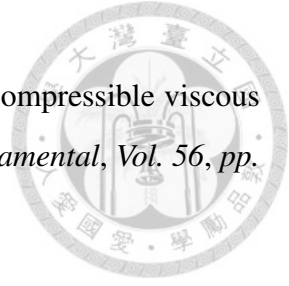
- 
- [39] Tony W. H. Sheu, S. F. Tsai, M. M. T. Wang, Discussion of numerical deficiency of applying a partial weighted upwind finite element model to incompressible Navier-Stokes equations, *Numerical Heat Transfer Part B : Fundamental*, Vol. 32, pp. 197-214, 1997.
- [40] P. H. Chiu, R. K. Lin, Tony W. H. Sheu, A differential interpolated direct forcing immersed boundary method for predicting incompressible Navier-Stokes equations in time-varying complex geometries, *Journal of Computational Physics*, Vol. 229, pp. 4476-4500, 2010.
- [41] R. M. Smith, A. G. Hutton, The numerical treatment of advection-a performance comparison of current methods, *Numerical Heat Transfer*, Vol. 5, pp. 439-461, 1982.
- [42] P. Tamamidis, D. N. Assanis, Evaluation for various high-order-accuracy scheme with and without flux limiters, *International Journal for Numerical Methods in Fluids*, Vol. 16, pp. 931-948, 1993.
- [43] C. Taylor, P. Hood, A numerical solution of the Navier-Stokes equations using the finite element technique, *Computer & Fluids*, Vol. 1, pp. 73-100, 1973.
- [44] R. L. Sani, P. M. Gresho, R. L. Lee, D. F. Griffiths, M. Engelman, The cause and cure (!) of the spurious pressure generated by certain FEM solutions of the incompressible Navier-Stokes equations: Part 2, *International Journal for Numerical Methods in Fluids*, Vol. 1, pp. 171-204, 1981.
- [45] O. Ladyzhenskaya, The mathematical theory of viscous incompressible flow, *Gordon and Breach, New York*, 1969.
- [46] F. Brezzi, On the existence, uniqueness and approximation of saddle point problems arising from Lagrangian multipliers, *RAIRO, Anal. Num.*, Vol 8(R2), pp. 129-151, 1974.
- [47] I. Babuska, Error bounds for finite element methods, *Numerische Mathematik*, Vol. 16, pp. 322-333, 1971.



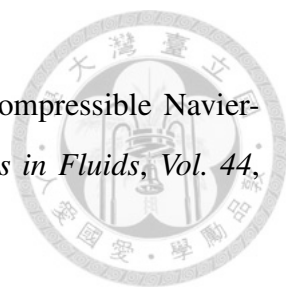
- [48] M. Bercovier, O. Pironneau, Error estimate for the finite element solution of the Stokes problem in the primitive variables, *Numerische Mathematik*, Vol. 33, pp. 211-244, 1979.
- [49] C. Taylor, T. G. Hughes, Finite element programming of the Navier-Stokes equations, *Pineridge Press Swansea*, 1981.
- [50] B. Irons, A frontal solution program for finite element analysis, *International Journal for Numerical Methods in Engineering*, Vol. 2, pp. 5-32, 1970.
- [51] P. H. Chiu, Tony W. H. Sheu, On the development of a dispersion-relation-preserving dual-compact upwind scheme for convection-diffusion equation, *Journal of Computational Physics*, Vol. 2009, pp. 3640-3655, 2009.
- [52] U. Ghia, K. N. Ghia, C. T. Shin, High-Re solutions for incompressible flow using the Navier-Stokes equations and a Multigrid method, *Journal of Computational Physics*, Vol. 48, pp. 387-411, 1982.
- [53] E. Etrurk, T. C. Corke, C. Gökcöl, Numerical solutions of 2-D steady incompressible driven cavity flow at high Reynolds numbers, *International Journal for Numerical Methods in Fluids*, Vol. 48, pp. 747-774, 2005.
- [54] T. Coupez, E. Hachem, Solution of high-Reynolds incompressible flow with stabilized finite element and adaptive anisotropic meshing, *Computer Methods in Applied Mechanics and Engineering*, Vol. 267, pp. 65-85, 2013.
- [55] J. P. Pontaza, J. N. Reddy, Space-time coupled spectral/hp least-squares finite element formulation for the incompressible Navier-Stokes equations, *Journal of Computational Physics*, Vol. 197, pp. 418-459, 2004.
- [56] L. D. Dailey, R. H. Pletcher, Evaluation of multigrid acceleration for preconditioned time-accurate Navier-Stokes algorithms, *Computer & Fluids*, Vol. 25 (8), pp. 791-811, 1996.

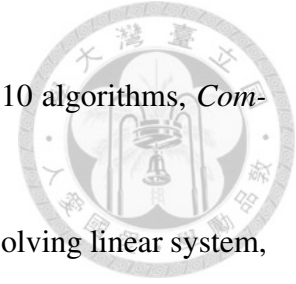


- [57] M. T. Wang, Tony W. H. Sheu, Implement of a free boundary condition to Navier-Stokes equations, *International Journal for Numerical Methods for Heat & Fluid Flow*, Vol. 7, pp. 95-111, 1997.
- [58] D. K. Gartling, A test problem for outflow boundary conditions flow over a backward-facing step, *International Journal for Numerical Methods in Fluids*, Vol. 11, pp. 953-967, 1990.
- [59] E. Erturk, Numerical solutions of 2-D steady incompressible flow over a backward-facing step, Part I: High Reynolds number solutions, *Computers & Fluids*, Vol. 37, pp. 633-655, 2008.
- [60] P. M. Gresho, D. K. Gartling, K. H. Winters, T. J. Garratt, A. Spence and J. W. Goodrich, Is the steady viscous incompressible two-dimensional flow over a backward-facing step at $Re = 800$ stable ? *International Journal for Numerical Methods in Fluids*, Vol. 17, pp. 501-541, 1993.
- [61] R. L. Sani and P. M. Gresho, Résumé and remarks on the open boundary condition minisymposium, *International Journal for Numerical Methods in Fluids*, Vol. 18, pp. 983-1008, 1994.
- [62] I. E. Barton, The entrance effect of laminar flow over a backward-facing step geometry, *International Journal for Numerical Methods in Fluids*, Vol. 25, pp. 633-644, 1997.
- [63] J. Keskar, D. A. Lyn, Computations of a laminar backward-facing step flow at $Re = 800$ with a spectral domain decomposition method, *International Journal for Numerical Methods in Fluids*, Vol. 29, pp. 411-427, 1999.
- [64] D. C. Wan, B. S. V. Patnaik, G. W. Wei, Discrete singular convolution finite sub-domain method for the solution of incompressible viscous flows, *Journal of Computational Physics*, Vol. 180, pp. 229-255, 2002.

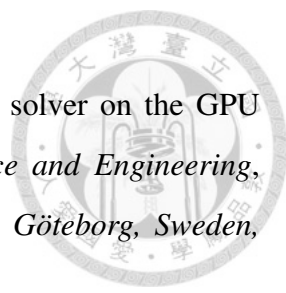



- [65] Tony W. H. Sheu, M. C. Hsu, Finite element simulation of incompressible viscous flow in moving meshes, *Numerical Heat Transfer, Part B : Fundamental*, Vol. 56, pp. 38-57, 2009.
- [66] S. Abide, S. Viazzo, A 2D compact fourth-order projection decomposition method, *Journal of Computational Physics*, Vol. 206, pp. 252-276, 2005.
- [67] M. Najafi, V. Enjilela, Natural convection heat transfer at high Rayleigh numbers- Extended meshless local Petrov-Galerkin (MLPG) primitive variable method, *Engineering Analysis with Boundary Elements*, Vol. 44, pp. 170-184, 2014.
- [68] J.C. Kalita, D.C. Dalal, A.K. Dass., Fully compact higher-order computation of steady-state natural convection in a square cavity, *Physics Review E*, Vol. 64, 066703, 2001.
- [69] Tony W. H. Sheu, P. H. Chiu, A divergence-free-condition compensated method for incompressible Navier-Stokes equations, *Computer Methods in Applied Mechanics and Engineering*, Vol. 196, pp. 4479-4494, 2007.
- [70] H. N. Dixit, V. Babu, Simulation of high Rayleigh number natural convection in a square cavity using the lattice Boltzmann method, *International Journal of Heat and Mass Transfer*, Vol. 49, pp. 727-739, 2006.
- [71] D. C. Wan, B. S. V. Patnaik, G. W. Wei, A new benchmark quality solution for the buoyancy-driven cavity by discrete singular convolution, *Numerical Heat Transfer Part B : Fundamental*, Vol. 40, pp. 199-228, 2001.
- [72] Y. Saad, Iterative methods for sparse linear system, *SIAM press*, 2003.
- [73] Tony W. H. Sheu, M. M. T. Wang, A comparison study on multivariate and univariate finite element for three-dimensional incompressible viscous flow, *International Journal for Numerical Methods in Fluids*, Vol. 21, pp. 683-696, 1995.

- 
- [74] Tony W. H. Sheu, R. K. Lin, Newton linearizaion of the incompressible Navier-Stokes equations, *International Journal for Numerical Methods in Fluids*, Vol. 44, pp. 297-312, 2004.
- [75] Neo S. C. Kao, Tony W. H. Sheu, S. F. Tsai, On a wavenumber optimized streamline upwinding method for solving steady incompressible Navier-Stokes equations, *Numerical Heat Transfer, Part B : Fundamental*, Vol. 67, pp. 1-25, 2015.
- [76] T. J. R. Hughes, I. Levit, J. Winget, Element-by-element implicit algorithm for heat conduction, *Journal of Engineering Mechanics*, Vol. 109 (2), pp. 576-585, 1983.
- [77] G. F. Carey, E. Barragy, R. McLay, M. Sharma, Element-by-element vector and parallel computations, *Communications in Applied Numerical Methods*, Vol. 4 (3), pp. 299-307, 1988.
- [78] Morten M. T. Wang, Tony W. H. Sheu, An element-by-element BICGSTAB iterative method for three-dimensional steady Navier-Stokes equations, *Journal of Computational and Applied Mathematics*, Vol. 79, pp. 147-165, 1997.
- [79] T. E. Tezduyar, Stabilized finite element formulations for incompressible flow computations, *Advances in applied mechanics*, Vol. 28, pp. 1-44, 1992.
- [80] K. K. Phoon, Iterative solution of large-scale consolidation and constrained finite element equations for 3D problems, *International e-Conference on Modern trends in Foundation Engineering : Geological Challenges and Solutions*, Indian Institute of Technology, Madras, India, Jan. 26-30, 2004.
- [81] C. C. Fang, Development of a finite element method for inviscid Euler equations, *Ph.D thesis, Department of Naval Architecture and Ocean Engineering, Taipei, Taiwan*.
- [82] M. T. Heath, Scientific Computing : An introductory survey, *McGraw Hill press*, 2005.



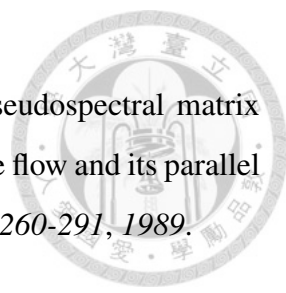
- [83] J. Dongarra, F. Sullivan, Guest editor's introduction to the top10 algorithms, *Computing in Science and Engineering*, Vol. 2(1), pp. 22-23, 2000.
- [84] M. R. Hestenes, E. Stiefel, Methods of conjugate gradient for solving linear system, *Journal of research of the National Bureau of Standard*, Vol. 49, No. 6, pp. 409-436, 1952.
- [85] R. Fletcher, Conjugate Gradient methods for indefinite system, *Lecture notes in Math.* 506, pp. 73-89, 1976.
- [86] H. A. Van der Vorst, BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear system *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, pp. 631-644, 1992.
- [87] Y. Saad, M. H. Schultz, GMRES : A generalized minimal residual algorithm for solving nonsymmetric linear system, *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, pp. 856-869, 1996.
- [88] C. C. Fang, Tony W. H. Sheu, Two element-by-element iterative solutions for shallow water equations, *SIAM Journal on Science Computing*, Vol. 22, pp. 2075-2092, 2001.
- [89] H. Ding, C. Shu, K. S. Yeo, D. Xu, Numerical computation of three-dimensional incompressible viscous flows in the primitive variable form by local multiquadratic differential quadrature method, *Computer Methods in Applied Mechanics and Engineering*, Vol. 195, pp. 516-533, 2006.
- [90] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, A Survey of General-Purpose Computation on Graphics Hardware, *Computer Graphics Forum*, Vol. 26, No. 1, pp. 80-113, 2007.
- [91] NVIDIA, NVIDIA CUDA Compute Unified Device Architecture Programming Guide, Version 7.0, 2014.

- 
- [92] Niklas Karlsson, An incompressible Navier-Stokes equations solver on the GPU using CUDA, *Master thesis, Department of Computer Science and Engineering, Chalmers University of Technology, University of Gothenburg, Göteborg, Sweden, August 2013.*
- [93] K. Murano, T. Shimobaba, A. Sugiyama, N. Takada, T. Kakue, M. Oikawa, T. Ito, Fast computation of computer-generated hologram using Xeon Phi coprocessor, *Computer Physics Communications, Vol. 185, pp. 2742-2757, 2014.*
- [94] David B. Kirk, Wen-mei W. Hwu, Programming Massively Parallel Processors: A Hands-on approach, 2nd edition, *Elsevier press, 30 Corporate Drive, Suit 400, Burlington, MA 01803, USA, 2012.*
- [95] NVIDIA Whitepaper, NVIDIA's Next Generation CUDATM Compute Architecture: Kepler GK110, *Version 1.0, 2012.*
- [96] The Portland Group, CUDA Fortran programming guide and reference release, *2014.*
- [97] G. Ruetsch, M. Fatica, CUDA Fortran for Scientists and Engineers, *NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA 95050, 2011.*
- [98] B. C. Lee, R. W. Vudue, J. W. Demmel, K. A. Yelik, Performance models for evaluating and automatic tuning of symmetric sparse matrix-vector multiply, *ICPP'04. Proceedings of the 2004 International Conference on parallel processing, IEEE computer society, pp. 169-176, 2004.*
- [99] N. Bell, M. Garland, Efficient sparse matrix-vector multiplication on CUDA, *NVIDIA Technical Report, NVR-2008-004, NVIDIA Corporation, Dec. 2008.*
- [100] M. M. Baskaran, R. Bordawekar, Optimization sparse matrix-vector multiplication on GPU, *IBM Research Report, RC24704, April, 2009.*

- 
- [101] S. Williams, N. Bell, J. W. Choi, M. Garland, L. Oliker, R. Vuduc, Scientific computing with multicore and accelerators, *IBM Research Report, CRC press, 2010.*
- [102] R. Li, Y. Saad, GPU-accelerated preconditioned iterative linear solver, *The Journal of Supercomputing, Vol. 63(2), pp. 443-466, 2012.*
- [103] A. Gaikwad, I.M. Toke, Parallel iterative linear solvers on GPU: A financial engineering case, *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Newwork-based Processing, DOI 10.1109/PDP.2010.55, pp. 607-614, Washington DC, USA, IEEE Computer Society, 2010.*
- [104] M. Ament, G. Knittel, D. Weiskopf, W. Strasser, A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform, *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Newwork-based Processing, DOI 10.1109/PDP.2010.51, pp. 583-592, 2010.*
- [105] I. Kiss, S. Gyimóthy, Z. Badics, J. Póvó, Parallel realization of the element-by-element FEM technique by CUDA, *IEEE Transactions of Magnetics, Vol. 48(2), pp. 507-510, 2012.*
- [106] D. Komatitsch, D. Michéa, G. Erlebacher, Porting a high-order finite element earthquake modeling application to NVIDIA graphics cards using CUDA, *Journal of Parallel and Distributed Computing, Vol. 69, pp. 451-460, 2009.*
- [107] J. Eustice, Flow of water in curved pipe, *Proceeding of the Royal Society of London Series A, Vol. 84, pp. 107-118, 1910.*
- [108] J. Eustice, Experiments on stream-line motion in curved pipes, *Proceeding of the Royal Society of London Series A, Vol. 85, pp. 119-131, 1911.*
- [109] W. R. Dean, The stream-line motion of fluid in a curved pipe, *Philosophie Magazin, Vol. 20, pp. 208-223, 1927.*



- [110] R. K. Shah, A. L. London, Laminar flow forced convection in ducts, *Academic press, New York, San Fransisco, London* , 1978.
- [111] J. A. C. Humphrey, A. M. K. Taylor, J. H. Whitelaw, Laminar flow in a square duct of strong curvature, *Journal of Fluid Mechanics, Vol. 83, pp. 509-527, 1977.*
- [112] F. Sotiropoulos, W. J. Kim, V. C. Patel, A computational comparison of two incompressible Navier-Stokes solvers in three-dimensional laminar flows, *Computer & Fluids, Vol. 23(4), pp. 627-646, 1994.*
- [113] Y. Argawal, L. Talbot, K. Gong, Laser anemometer study of flow development in curved circular pipe, *Journal of Fluid Mechanics, Vol. 85, pp. 497-518, 1978.*
- [114] W. Y. Soh, S. A. Berger, Laminar entrance flow in a curved pipe, *Journal of Fluid Mechanics, Vol. 148, pp. 109-135, 1984.*
- [115] I. E. Barton, A numerical study of flow over a confined backward-facing step, *International Journal for Numerical Methods in Fluids, Vol. 21, pp. 653-665, 1995.*
- [116] G. Biswas, M. Breuer, F. Durst, Backward-facing step flows for various expansion ratios at low and moderate Reynolds number, *Journal of Fluid Engineering, Vol. 126, pp. 362-374, 2004.*
- [117] M. A. Cruchaga, A study of the backward-facing step problem using generalized streamline formulation., *Communications in Numerical Methods in Engineering, Vol. 14, pp. 697-708, 1998.*
- [118] T. C. Papanastasiou, N. Malamataris, K. A. Ellwood, A new outflow boundary condition, *International Journal for Numerical Methods in Fluids, Vol. 14, pp. 587-608, 1992.*
- [119] B. F. Armaly, F. Durst, J. C. F. Pereira, B. Schönung, Experimental and theoretical investigation of backward-facing step flow, *Journal of Fluid Mechanics, Vol. 127, pp. 473-496, 1983.*

- 
- [120] H. C. Ku, R. S. Hirsh, T. D. Taylor, A. P. Rosenberg, A pseudospectral matrix element method for solution of three-dimensional incompressible flow and its parallel implementation, *Journal of Computational Physics*, Vol. 83, pp. 260-291, 1989.
- [121] B. N. Jiang, L. J. Hou, T. L. Lin, L. A. Povinelli, Least-squares finite element solution for three-dimensional backward-facing step flow, *International Journal of Computational Fluid Dynamics*, Vol. 4, pp. 1-19, 1995.
- [122] P. T. William, A. J. Baker, Numerical simulations of laminar flow over a 3D backward-facing step, *International Journal for Numerical Methods in Fluids*, Vol. 24, pp. 1159-1183, 1997.