國立交通大學

土木工程學系

碩士論文

CPU 平行粒子群最佳化應用於平面桁架結構最佳化設計

Application of Particle Swarm Optimization using CPU Parallel Processing in Optimization Design of 2D Truss Structures

研 究 生 ：洪銘澤

指導教授：洪士林　博士

中華民國　一〇二年七月

CPU 平行粒子群最佳化應用於桁架結構最佳化設計

Application of Particle Swarm Optimization using CPU Parallel Processing in Optimization Design of 2D Truss Structures

研 究 生：洪銘澤　　Student：Ming-Tse Hung

指導教授：洪士林　　Advisor：Dr. Shih-Lin Hung

國 立 交 通 大 學

土 木 工 程 系

碩 士 論 文

A Thesis

Submitted to Department of Civil Engineering

College of Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Civil Engineering

July 2013

Hsinchu, Taiwan, Republic of China

中華民國一〇二年七月

CPU 平行粒子群最佳化應用於桁架結構最佳化設計

研究生：洪銘澤　　　　　　　　　　　指導教授：洪士林 博士

國立交通大學土木工程學系碩士班

摘要

電腦硬體不斷進步，電腦從以前單核心到現在多核心處理器。核心時脈上升幅度逐漸趨緩，因此使用平行運算才能妥善利用多核心處理器。粒子群最佳化(Particle Swarm Optimization, PSO)是一個在最佳化中最常被使用的方法之一。它是模擬群體智慧(Swarm Intelligent)的仿生演算法，先以全域搜尋方式在搜尋範圍中蒐集資訊，搜尋結果較好的粒子趨使其他的粒子靠近轉為局部搜尋，最後收斂到粒子群經驗中最佳解。由於平行程式適合資料分別獨立運算，盡可能避免執行緒同步才能有更好的效能。本研究先利用『OpenMP』平行運算矩陣乘法(Matrix Multiplication)。測試矩陣大小與平行化效能的關係。利用粒子群最佳化演算法，粒子分別獨立搜尋個體的特性平行運算，再分別用【30 個變數平方和】、【Beale's Function】、和【桁架最佳化設計】測試平行運算粒子群最佳化的效能和粒子總數的關係和效能和迭代次數的關係，最後附上最佳化結果。本研究結果指出，粒子群最佳化當迭代次數和總粒子數量夠多時。兩種演算法效能都可以達到加速比 3 以上的效果。此平行運算粒子群最佳化演算法可應用於其他最佳化案例只需更改目標函數。

關鍵字：OpenMP、平行運算、粒子群最佳化、平面桁架結構最佳化設計

Application of Particle Swarm Optimization using CPU Parallel Processing in Optimization Design of 2D Truss Structures

Student: Ming-Tse Hung　　　　　　　　Advisor：Dr. Shih-Lin Hung

Department of Civil Engineering

College of Engineering

National Chiao Tung University

Abstract

Advances in computer hardware, computers has been progressed from single-core to multi-core processors. This progress in computers provides a possibility of development in parallel computing. OpenMP, currently, is a popular API for implement parallel programs on PCs with C, C++, or FORTRAN. Among bionic optimization algorithms, particle swarm optimization (PSO) is a most employed approach for solving optimization problem due to the characteristics of less working parameters and quick convergence rate. The algorithm has been proved capable of reach near global optimization in search space using particle itself and swarm experience in search. Since computing for each single particle is independent in searching iteration, the PSO algorithm can be paralleled in a multi-core computing platform. The objective of this study is to develop a parallel PSO algorithm in optimization design of 2D truss structures. First, the problem of synchronization in shared memory platform for the algorithm is studied and solved. Following, a parallel matrix multiplication program with different size is developed and the computing performance related to matrix size is then assessed. Second, a parallel PSO program with OpenMP is implemented. A Sphere function with 30 dimensions and a Beale's function are utilized to verify and evaluate the correctness and the corresponding performance. Finally, three 2D truss optimization design problems are solved using the proposed parallel PSO program. Computing results reveal that the parallel program achieves a speedup factor greater than 3.42 under a four-core computing platform with OpenMP.

Keywords: OpenMP, parallel computing, Particle swarm optimization (PSO),

2D truss optimization

誌謝

首先感謝我的指導老師洪士林教授，老師常在我遇到瓶頸時幫忙我，在口試前也專門對我的簡報一頁一頁指導我修正簡報，使我受益良多，口試時也更加順利，感謝老師兩年來的教導。感謝黃炯憲老師、林昌佑老師和詹君治學長於百忙中撥空參加我的口試，並給我予許多寶貴意見。使我的論文能夠更為完整，在此也謝謝各位老師的教導。

感謝詹君治學長兩年來在簡報、口頭報告、和專業知識的指導。感謝江祥學長、勇奇學長、穎泰學長、思伶學姐、孟軒學長、宣治學長、晟佑學長、和俊佐學長給予我幫忙與鼓勵，江祥學長特別為了我跑到學校真的非常感謝。

感謝研究所同學子陽、錦鴻、奇霖、義洋、耀緯、明廉、冠龍、湘銘、曉德、和京陞，在研究所的日子裡一起研究和玩樂。感謝在清大 EPL 實驗室的宇廷和他的學弟妹們在我程式出現問題時給予幫助和建議。感謝學弟丁丁、智嵩、建文和允璿在我研究所後半日子中一起研究和玩樂。

感謝國中同學，每當我回臺北休息時一起出來打球、吃飯、和聊天，讓我分享在研究所發生的一些有趣事情也讓我抒發課業壓力。

在研究所生活中有許多貴人的幫助才能完成論文，在此感謝所有幫助過我的人，也希望幫助我的人往後一切順利。

最後將本篇論文獻給我的爸爸和媽媽，從小養育我到現在，因為有你們的奉獻、付出、栽培和支持，才能讓我在研究所生活順順利利無後顧之憂的完成學業，沒有你們就沒有今天的我。謝謝你們的養育之恩，我永遠都不會忘記的。

洪銘澤　謹誌

中華民國 102 年 7 月于國立交通大學

目錄

摘要 I
Abstract II
誌謝 III
目錄 V
表目錄 VIII
圖目錄 X
第一章 緒論 1
1.1 背景與動機 1
1.2 研究目的 3
1.3 研究流程 3
1.4 研究架構 5
第二章 文獻回顧 7
2.1 平行運算簡介 7
2.1.1 多核心歷史發展 7
2.1.2 多核心運算的發展 8
2.1.3 平行程式主流架構與對應編譯語言 9
2.1.4 費林分類法(Flynn's Taxonomy) 10
2.1.5 OpenMP 簡介 11

2.1.6 OpenMP 平行方式 12
2.2 粒子群最佳化 12
第三章 研究方法 16
3.1 OpenMP 程式開發 16
3.1.1 競爭危害(Race Condition) 19
3.2 平行運算—粒子群最佳化 19
3.3 桁架最佳化設計 21
3.3.1 結構最佳化設計方法 21
3.3.2 桁架結構分析 23
3.4 加速比 28
第四章 結果分析與探討 30
4.1 平行運算矩陣乘法 30
4.2 平行運算粒子群最佳化 31
4.2.1 30 個變數平方和之最小值 32
4.2.2 Beale's function 33
4.2.3 桁架斷面結構最佳化 33
第五章 結論與建議 37
5.1 結論 37
5.2 建議 38
參考文獻 39

附表 41

附圖 62

表目錄

表 1　費林分類法[17] 41

表 2　OpenMP Directives 42

表 3　矩陣乘法測試結果 42

表 4　30 個變數的最小平方和最佳化執行時間與加速比 43

表 5　30 個變數的最小平方和最佳化執行結果 44

表 6　修改平行程式後測試 45

表 7　Beale's function 粒子數與加速比關係 46

表 8　10 支桿件桁架斷面最佳化結果 47

表 9　10 支桿件最佳化執行時間與粒子總數關係 48

表 10　10 支桿件最佳化和迭代次數關係 49

表 11　10 支桿件最佳化斷面後節點位移 50

表 12　10 支桿件最佳化斷面後結構分析桿件應力 51

表 13　17 支桿件桁架斷面最佳化結果 52

表 14　17 支桿件最佳化和粒子總數關係 53

表 15　17 支桁架結構最佳化斷面後節點位移 54

表 16　17 支桁架結構最佳化斷面後結構分析桿件應力 55

表 17 26 支桁架結構桿件編號表 56

表 18 26 支桁架結構最佳化和粒子總數關係 57

表 19 26 支桁架結構最佳化斷面後桿件應力 58

表 20 26 支桁架結構最佳化斷面後節點位移 59

表 21 26 支桁架結構最佳化斷面結果 60

圖目錄

圖 1 傳統連續處理 62

圖 2 平行處理 62

圖 3 平行層次圖[18] 63

圖 4 OpenMP 發展流程圖 63

圖 5 Fork/Join Model 64

圖 6 粒子群最佳化粒子速度和位置更新示意圖 64

圖 7 粒子群最佳化流程圖 65

圖 8 簡易 OpenMP 程式 66

圖 9 簡易 OpenMP 程式執行結果 66

圖 10 簡易 OpenMP 程式示意圖 67

圖 11 矩陣乘法 68

圖 12 矩陣乘法使用 OpenMP 68

圖 13 競爭危害範例 69

圖 14 競爭危害輸出結果 69

圖 15 解決競爭危害 70

圖 16 粒子群最佳化虛擬碼 71

圖 17 二維桁架桿件局部座標和整體座標......72

圖 18 矩陣乘法執行效能結果......73

圖 19 核心數與加速比關係......73

圖 20 多執行緒同時存取同一個記憶體位置......74

圖 21 平行粒子群最佳化演算法......76

圖 22 粒子群最佳化核心......76

圖 23 30 個變數的最小平方和粒子總數與加速比關係......77

圖 24 4.2.1 節最佳化收斂情形......78

圖 25 Beale's function 繪製圖形......78

圖 26 Beale's function 粒子總數與加速比關係......79

圖 27 Beale's function 收斂情形......79

圖 28 10 支桿件桁架結構......80

圖 29 10 支桁架加速比與粒子總數關係......81

圖 30 10 支桿件桁架結構最佳化收斂過程......82

圖 31 17 支桁架結構......82

圖 32 17 支桁架加速比與粒子總數關係......83

圖 33 17 支桿件桁架結構最佳化收斂過程......83

圖 34 26 支桁架結構......84

圖 35　26 支桁架結構粒子總數與加速比關係 84

圖 36　26 支桁架結構收斂情形 85

第一章 緒論

1.1 背景與動機

隨著電腦的中央處理器(CPU)，時脈(Hz)上升速度逐漸趨緩，然而核心(core)已經從單核心變成多核心，甚至手機的處理器也是多核心。 在一般在撰寫程式都以單核心做運算。如圖 1 所示，而其他核心則處於閒置(idle)狀態，因此盡可能地將程式平行化才能將硬體有效利用，如圖 2 所示。

平行電腦架構一般分為分享記憶體(Shared Memory Multiprocessor)、分散記憶體(Distributed Memory Multicomputer)、和混合式(Hybrid Distributed-Shared Memory)。本研究以分享記憶體(Shared Memory Multiprocessor)為主，利用應用程式『OpenMP』開發平行程式[1]。

撰寫平行程式時必須先考量程式中哪些程序可以平行，例如矩陣相乘因為在相乘時每個元素均是獨立的個體並不會互相影響。若沒有處理平行程式容易造成競爭危害(race condition)或鎖死(dead locked)等情形可能造成效能不佳，甚至造成資料錯誤。

平行程式最大的優點可以縮短計算時間解決問題或解決更大的問題。因此本研究以計算量大的最佳化(Optimization)為主，而最佳化是指在許多限制與條件中，尋找最佳答案的過程，而最佳答案代表最好的妥協。而平行化則可得到更好的“最好的妥協”，例如時間條件或答案限制。最佳化在早期多以梯度做為搜尋方向，當求解極值問題有很好的效率。因此當初始點位在全域最佳解梯度附近維持一致處可以快速搜尋到全域最佳值。反之若位在區域最佳解附近梯度維持一致處則會掉入區域最佳解[2]。因此許多學者為了改善傳統梯度法容易掉入區域最佳解缺點而開發出不需要利用梯度的最佳化演算法，即非傳統最佳化演算法。其中包含：粒子群最佳化(Particle Swarm Optimization) [7]、蟻群演算法(Ant Colony Optimization) [15]、模擬退火演算法(Simulated Annealing) [20]、調和搜尋演算法(Harmony Search) [16]、和禁制搜尋法(Tabu Search)[21]，上述方法都改善了傳統梯度搜尋法的缺點擁有更好的全域搜尋能力。

桁架結構是在土木工程常見的結構系統之一，桁架在理論中只以軸力傳遞，因此在做結構分析時只需考慮軸力即可。在做結構分析時，桁架結構較其他結構系統簡單。桁架最佳化一般分為三種，最佳桿件斷面尺寸設計(Size optimization)、最佳結構配置(Configuration optimization)、和最佳拓

樸設計(Topology optimization)。最佳桿件斷面尺寸設計最為普遍使用[3]。

1.2 研究目的

平行程式最大的優點可以縮短計算時間解決問題或解決更大的問題。由於粒子群最佳化在搜尋最佳解以全域搜尋需要大量的迭代次數和大量的粒子數增加其搜尋能力，因此粒子群最佳化耗費大量的計算資源。

本研究目的希望利用現今平行運算技術(OpenMP)及硬體架構發展一套有效的演算法。粒子群最佳化中粒子分別獨立之特性將此最佳化程式中計算粒子適應性函數時平行化減少計算時間。最佳化測試案例分別有：

- $\min F(x_0, x_1, \ldots, x_{29}, x_{30}) = x_0^2 + x_1^2 + \ldots x_{29}^2 + x_{30}^2$
- 桁架斷面最佳化

1.3 研究流程

本研究使用 OpenMP 平行處理粒子群最佳化桿件斷面尺寸設計流程一共分為以下步驟：

第一步驟：

蒐集粒子群最佳化相關文獻並且了解粒子群最佳化理論和搜尋最佳化方法。蒐集 OpenMP 相關教學自我學習。了解多執行緒運作方式和如何使用共享記憶體。

第二步驟：

參考網路教學文章編寫平行程式矩陣乘法，並且計算運算時間得到程式加速比。

第三步驟：

參考文獻[7]編寫粒子群最佳化程式。利用簡單數學最佳化方程式驗證演算法是否正確。

第四步驟：

編寫結構分析程式。桁架結構桿件應力和節點位移作為設計依據。

第五步驟：

將第三步驟和第四步驟兩程式結合。第四步驟為目標函數並且當不符合設計限制時增加懲罰函數，分析結束後回傳桁架結構總重。

第六步驟：

將 OpenMP 指令加入第五步驟程式中，並且增加計算所需資料副本，確保多執行緒可以同時運算。

第七步驟：

改變粒子總數、迭代次數、和不同目標函數分別計算程式加速比。

第八步驟：

平行粒子群最佳化程式開發完成，並提出建議與未來展望。

1.4 研究架構

本研究論文共分五個章節：

第一章緒論：說明研究背景、動機、目的、和研究流程。

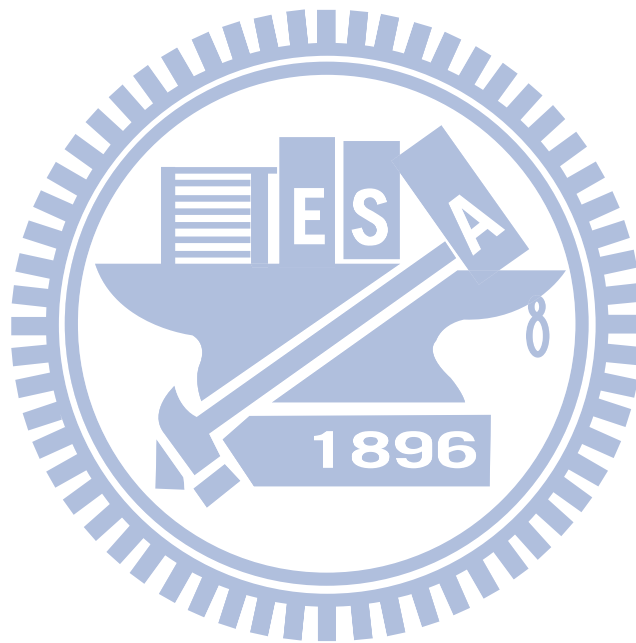
第二章文獻回顧：首先介紹多核心架構和歷史，架構所對應的編譯語言；費林分類法(Flynn's Taxonomy)分別介紹 SISD、SIMD、MISD、和 MIMD 四種高效能電腦的分類方式。『OpenMP』的起源並且簡單介紹其編寫方式，如何使用『OpenMP』控制多執行緒達成平行運算；簡單介紹粒子群最佳化歷史和理論。

第三章研究方法：更深入介紹使用『OpenMP』編寫平行程式，並且利用自身發現的錯誤情形介紹平行程式編寫時常發生競爭危害(Race Condition)。實際將粒子群最佳化演算法說明，並且提出應該平行化的演算法區段；介紹桁架結構分析和最佳化的目標函數和限制條件。

第四章結果分析與探討：先利用一般人熟悉矩陣乘法，實際測試平行

化結果。粒子群最佳化的平行運算。先說明平行化演算法流程，利用【30個變數平方和之最小值】和【桁架斷面結構最佳化】測試平行化效果。

第五章結論與建議：為整篇論問下結論並且提出未來可以繼續發展的目標。



第二章 文獻回顧

2.1 平行運算簡介

2.1.1 多核心歷史發展

傳統電腦軟體以串行計算(serial computation)方式撰寫，一個指令執行完成才能執行下一個指令[1]。平行計算一般是指許多指令得以同時進行的計算模式。在同時進行的前提下，可以將計算的過程分解成小部份，之後以平行方式來加以解決[4]。平行運算早在二十世紀 60 年代美國就有平行計算機之後 70、和 90 年代皆有平行計算機的蹤跡。過去處理器依莫爾定律[5]所述，每十八個月增加一倍的電晶體數逐漸遞增。處理器時脈增加的速度超越建構平行電腦和編譯平行程式的時間，使得平行電腦架構與編譯平行程式效益有限且很快地就因處理器時脈增加而被取代。

近年處理器逐漸從單晶片單核心到單晶片多核心發展。因電路設計的物理極限、散熱、和漏電(這裡的漏電指高製程晶片於運作時，部分電子因量子穿隧效應而由非電路或電路元件的矽中穿過)等原因使得處理器往單晶片多核心發展，在 2005 年多核心架構開始量產銷售。多核心處理器的出現讓平行運算有更好的效益，軟體逐漸開始配合硬體而往平行運算發展。

平行運算效益提高後各領域開始運用於科學計算包括奈米科學、工程計算、模擬爆破、氣象、醫學影像、流體力學、模擬粒子碰撞、石油探勘、和材料科學等應用。一般應用於影音轉檔、影音播放、遊戲物理效果、解壓縮工具、繪圖工具、和瀏覽器等。現今的平行運算已經不再侷限於某些程式，越來越多日常用到的應用程式漸漸以平行運算增加其效能。

2.1.2 多核心運算的發展

平行運算是一個廣義的概念，根據平行架構不同一般可分為以下幾種方式。如圖 3 所示最微觀的平行運算是單核心指令平行(ILP-Instruction Level Parallelism)，使用單處理器的執行單元同時執行多條指令。**多核心平行**(Muti-Core Parallel)即在單晶片擁有多個核心，也就是將執行緒平行(TLP-Thread Level Parallelism)。多處理器平行(Multi-Processor Parallel)，即安裝多個處理器在同一機器上，以執行緒與處理過程(process)等級的平行。最後藉由網路連結多台電腦實現叢集(Cluster)或分散式(Distributed)平行。

隨著平行架構的發展，平行演算法(Parallel Algorithm)也不斷成熟，歷史上平行演算法研究集中於二十世紀七八十年代。這段時間出現許多不同

互連架構和記憶體模式的 SIMD(Single Instruction Multiple Data)計算機，到了二十世紀九十年代中期，平行演算法更精進其設計與分析也兼顧平行電腦架構與軟體上設計。

2.1.3 平行程式主流架構與對應編譯語言

單晶片多核心：

2005 年起 Intel 與 AMD 兩大 CPU 製造商正式量產銷售雙核心處理器，2007 年與 2009 年又分別推出四核心與八核心處理器，共享記憶體(Shared Memory)的架構已經廣泛使用。開發者也開始發展多執行緒編程，利用多執行緒編程即可在多個核心使執行緒平行充分利用 CPU 的運算能力。多執行緒編程除了能利用作業系統本身提供 API，還可利用一些函式庫或語言擴充等實現多執行平行運算，而一般常見的是 intel 所發展的 OpenMP。

超級電腦、叢集和分散式運算

超級電腦是指效能在世界排名非常優異的電腦可以處理大量資料和高速運算，如美國 ORNL 實驗室的『泰坦(Titan)』、美國 IBM 的『Blue Gene/Q』、和日本 Fujitsu 的『京』。基本組成元件與個人電腦的概念無太大差異，但規格及效能較為強大，例如『泰坦(Titan)』的計算效能高達 17.59

Peta-FLOPS。計算機叢集簡稱叢集，它是透過一組鬆散整合的電腦軟體和硬體連起來高密度合作運算工作。叢集中每一台計算機稱作節點，通常利用區域網路連結。叢集在性能價格比上較超級電腦優異，目前超級電腦、叢集與分散式計算常用的開發工具是 MPI(Message Passing Interface)。

2.1.4 費林分類法(Flynn's Taxonomy)

費林分類法是一種高效能電腦的分類方式。1972 年費林(Michael J. Flynn)根據資料流(Information Stream)，分成指令(Instruction)和資料(Data)兩種，據此又可分為以下四種(表 1)：

- 單一指令流單一資料流，Single Instruction Single Data(SISD)。傳統計算機單 CPU 在同一時間只能執行一條指令處理一筆資料，即一個控制流和一個資料流順序執行。

- 單一指令流多資料流，Single Instruction Multiple Data(SIMD)。一個指令同時處理多個資料，即一個控制流多個資料流。

- 多指令流單一資料流，Multiple Instruction Single Data(MISD)。此系統結構只適用特定演算法，各處理器排成線性陣列，對應同一資料流並執行不同指令。

- 多指令流多資料流，Multiple Instruction Multiple Data(MIMD)。一般超級電腦或伺服器大多採用 MIMD 架構，多處理器處理不同指令流。

2.1.5 OpenMP 簡介

90 年代初期，提供共享記憶體(Shared Memory)的供應商提供利用 Fortran 以指令為基礎的擴充物件，使用者只需要增加指令在要平行的迴圈中即可。然而在 1994 年美國國家標準局 ANSI(American National Standards Institute) X3H5 第一次嘗試標準化 OpenMP，但分散式記憶體架構當時正流行因此未通過。1997 年 OpenMP 才制定標準化，至今持續在發展如圖 4[6]。

OpenMP(**Open** specification for **M**ulti-**P**rocessing)是一個應用程式介面(API)，用於控制多執行緒(multi-thread)和共享記憶體(shared memory)平行化。主要包含三個應用程式介面[6]：

- Compiler Directive
- Runtime Library Routines
- Environment Variables

多核心電腦已經非常普遍，但是傳統循序程式只能以單核心運作並無法發揮多核心的效能。將傳統循序程式由單一執行緒改寫為多執行緒才能發揮多核心的效能。撰寫多執行緒程式有很多種方法，除了利用作業系統本身提供 API 之外使用 OpenMP 是不錯的選擇。若只是簡單平行迴圈，程式設計師只需要將欲平行的區段加上 OpenMP 的指令即可將原本單執行緒的程式轉為多執行緒，效能也就大幅上升。程式設計師須注意程式的邏輯是否符合多執行緒，並且再適當時候做修正，並免造成競爭危害(Race Condition)、死結(Deadlock)。

2.1.6 OpenMP 平行方式

OpenMP 是以 Compiler Directive 為基礎，編譯器負責處理執行緒 fork/join 如圖 5 和分配任務給執行緒。共用記憶體架構配置方式一般分為動態和靜態主執行緒(master thread)在平行區域(Parallel region)前『fork』啟動其他執行緒(threads)，主執行緒再『join』等待所有執行緒執行完畢，之後主執行緒繼續以單執行緒方式執行。

2.2 粒子群最佳化

1995 年 Eberhart 和 Kennedy 提出粒子群最佳化(Particle Swarm Optimization, PSO)，它是一種具有群體智慧概念的演算法。粒子群最佳化起初因為兩學者 Eberhart 和 Kennedy 觀察鳥類覓食行為所得到靈感。將此

現象模擬成一套最佳化演算法，建立每個個體之間的互動規則產生群體行為而得到最佳化的目標[7]。

粒子群最佳化主要概念是假設有一群鳥在一個空間內尋找食物。初始並不知道最佳覓食位置在何處，所以每隻鳥則憑著各自經驗或直覺尋找地點，直到牠們覺得較佳的位置覓食。當某些鳥發現更好的覓食地點而去覓食，其他鳥就會收到訊息而往更好的地點前進覓食[8]。

粒子群最佳化演算法的模擬理論中，將『粒子(particle)』定義為解空間中的一隻鳥，而每個粒子其中都應包含：

- 位置(position)
- 速度(velocity)
- 個體經歷過最佳目標函數值(pbest)
- 群體經歷過最佳目標函數值(gbest)

每個粒子所在『位置』都是最佳化問題的解，每個解都對應一個答案稱之為『目標函數值(Objective function value)』或稱『適應值(Fitness

value)』[7]。粒子透過自己的『位置』、『pbest』、和『gbest』作為更新『速度』的依據。再將粒子原『位置』加上『速度』得到新的『位置』，計算新的『目標函數值』並與『pbest』比較。若較好則取代先前『pbest』，反之粒子回到原來『位置』。當所有粒子都更新完後，比較所有粒子『目標函數值』後，最好的粒子則為『gbest』，利用這樣的動作在解空間內反覆迭代搜尋最佳解。將以上最佳化方式利用數學式表達，假設一總粒子數 M、維度 N 個的最佳化問題，則『速度』與『位置』的更新如(式 1)和(式 2)

$$V_{i,j}^{k+1} = V_{i,j}^{k} + c_1 r_1 \left(P_{i,j}^{k} - X_{i,j}^{k}\right) + c_2 r_2 \left(P_{g,j}^{k} - X_{i,j}^{k}\right) \quad (1)$$

$$X_{i,j}^{k+1} = X_{i,j}^{k} + V_{i,j}^{k+1} \quad (2)$$

其中：

i： 從 1 到 M 的整數

j： 從 1 到 N 的整數

k： 迭代次數

$V_{i,j}^{k}$： 第i個粒子中第j個維度的速度向量

$X_{i,j}^{k}$： 第i個粒子中第j個維度的位置向量

$P_{i,j}^{k}$：　第i個粒子中第j個維度經歷過的最佳位置向量

$P_{g,j}^{k}$：　所有粒子經歷過第j個維度經歷過的最佳位置向量

c_1、c_2：學習因子

r_1、r_2：[0, 1]之間的隨機均勻亂數

學習因子c_1和c_2代表粒子移動趨勢之權重分別往每個粒子各自的最佳解和所有粒子的最佳解。當學習因子偏低代表粒子往目前各自最佳解和群體最佳解的趨勢較弱。利用前一次的速度向量徘徊搜尋，這個方式增加搜尋全域最佳解的機率，然而會增加最佳化的計算量與時間。相反的學習因子偏高代表粒子往前各自最佳解和群體最佳解的趨勢較強。可以減少粒子在搜尋區域中徘徊減少計算量與迭代次數，因此收斂速度較快，但也降低搜尋全域最佳解的機率。根據文獻[7]和[9]建議將學習因子設為常數 2 最適合。利用向量概念粒子的速度和位置更新如圖 6 所示。

第三章 研究方法

3.1 OpenMP 程式開發

『OpenMP』使用 Compiler Directive 的方式讓開發者撰寫平行程式，而非以函式庫的方式提供。開發者可以在 C/C++或 Fortran 的程式碼中加入 Compiler Directive。將程式可平行部分標示 Parallel region，分享變數標示 shared，獨立變數標示 private，或是分配資料方式等。修改完成後，將帶有 Compiler Directive 的程式碼交由支援『OpenMP』的編譯器進行編譯即為平行程式。[10]

因為『OpenMP』利用 Compiler Directive 的格式設計。若使用不支援『OpenMP』的編譯器去編譯包含 OpenMP Directive 編譯器將自動忽略產生原本循序程式，也可能因為編寫平行程式方式與循序程式不同造成資料錯誤或效能不佳。

本研究使用 C/C++編寫程式，OpenMP Directive 在 C/C++格式如表 2 所示

範例：

```
#pragma omp parallel default(None) private(x, y)
```

每個 OpenMP Directive 的開頭都是 #pragma omp，緊接著 Directive 指令將原本循序程式以多執行緒執行。

撰寫『OpenMP』平行程式與傳統循序程式不盡相同，因此首先必須瞭解執行緒(threads)的概念，以 C/C++為例利用簡單的程式做說明，如圖 6 所示，

第 1 行：

首先 C/C++程式必須載入『OpenMP』標頭檔，才能使用 OpenMP Directive、Runtime Library Routines、Environment Variables。

第 4 行：

標示 parallel region，parallel region 範圍內以多執行緒執行，此處未指定執行緒數量則以預設值為準。

第 6 行：

使用『OpenMP』中的 Runtime Library Routines，使用 `omp_get_thread_num()` 回傳目前使用的執行緒 ID。

第 10 行：

結束 parallel region，此『}』後，程式循序執行。

執行結果如圖 7 可知在 parallel region 中每個執行緒都會執行一次 parallel region 中的程式如圖 8，且並非按照執行緒編號執行，而是先執行完畢先輸出。因此在編寫平行程式時必須非常小心避免造成競爭危害(race

condition)。

了解執行緒的基本概念後利用矩陣乘法深入說明平行程式中那些變數可以分享(shared)那些需要私有(private)。一般循序編寫矩陣乘法如圖 9，其中矩陣 ArrA 和 ArrB 為已知數值。rowA 和 colB 為矩陣 ArrA 的列數和行數，colB 為矩陣 ArrB 的行數，經由三層槽狀迴圈執行矩陣乘法，存到矩陣 ArrC。使用『OpenMP』編寫平行運算矩陣乘法如圖 10，

第 2 行：

標示 parallel region，parallel region 範圍內以多執行緒執行，『OpenMP』預設變數為分享(shared)為預設值，default(none)將變數透過手動設定。

第 3 行：

將 ArrA, ArrB, 和 ArrC 三個變數設為共享(shared)，執行緒可以同時存取因此達到平行運算的效果。

第 4 行：

將槽狀迴圈內 i, j, k 三個變數設為私有(private)。避免多個執行緒修改，使每個 for 迴圈徹底執行完畢，否則造成會競爭危害(race condition)，使矩陣乘法錯誤。

第 6 行：

pragma omp for 為 OpenMP Directive 即告訴 compiler 此 for 迴圈以多執行緒執行。

比較圖 9 和圖 10 可以發現利用『OpenMP』將 for 迴圈以多執行緒執行非常容易僅需要增加一行指令，而迴圈內變數應共享(shared)或私有(private)，才是編寫『OpenMP』平行程式最重要的問題之一。

3.1.1 競爭危害(Race Condition)

利用一個簡單的程式來解釋造成競爭危害(race condition)的原因如圖 11，明顯看出這是兩個 for 迴圈中包一個加法。理論上此加法應執行十六次。輸出結果如圖 12，令人意外的是加法執行次數僅七次，且經多次執行此程式加法執行次數都不盡相同。造成此現象主要原因在於『OpenMP』會將在 parallel region 外的變數預設為共享(shared)變數。因此所有執行緒同時修改到相同的 j，導致執行次數比預期中得少。改善此情形需要將變數 j 設為私有(private)如圖 13 即可讓程式如預期執行十六次加法。

3.2 平行運算—粒子群最佳化

粒子群最佳化流程圖如圖 7，利用向量的概念讓粒子朝著最佳解搜尋。由圖 7、(式 1)、和(式 2)可知粒子更新位置和速度僅與粒子本身慣性

速度$V_{i,j}^{k}$、個別經歷最佳位置$P_{i,j}^{k}$、和群體經歷最佳位置$P_{g,j}^{k}$有關。因此每個粒子之間分別獨立並不會相互交換資料，正好符合平行運算的特性。

本研究利用『OpenMP』將粒子群最佳化以多執行緒達到平行運算效果。編寫時開始與循序程式大致相同以單執行緒的方式分別初始化粒子速度$V_{i,j}^{0}$、位置$X_{i,j}^{0}$、個別經歷最佳位置$P_{i,j}^{0}$、族群經歷最佳位置$P_{g,j}^{0}$，循序程式運作如圖 16 執行，可與流程圖(圖 7)對照參考，其中

第 8 行：

$f(P_g^0)$： 族群經歷最佳值

$f(X_i^0)$： 粒子i的目標函數值

第 11 行：

進入迭代，開始搜尋最佳解

第 19-22 行：

更新粒子i最佳位置$P_{i,j}^{k+1}$

第 24-27 行：

更新族群最佳位置$P_{g,j}^{k+1}$

第 28、29 行：

紀錄迭代次數；判斷是否結束迭代

當 do-while 迴圈符合停止條件時即得到最佳解$P_{g,j}^{k}$。

由圖 16 可知程式中 do-while 迴圈消耗最多運算資源，原因在於粒子群最佳化需要大量的迭代次數才能得到可接受的最佳解。本研究專注於 do-while 迴圈中平行運算。首先必須將已知資料複製四份到記憶體中後，將圖 16 第 13 行中【粒子總數】分為四等分。利用『OpenMP』complier directives，由執行緒 0 到 3 共四個執行緒分別執行第 14 行到第 22 行。在第 23 行離開 parallel region 即『OpenMP』中 join，以主執行緒來尋找此次迭代中最佳解位置$P_{g,j}^{k+1}$，反覆迭代後得到最佳解$P_{g,j}^{k}$。

3.3 桁架最佳化設計

3.3.1 結構最佳化設計方法

傳統上工程設計必須由有合格證照的工程師依照規範和其他相關規定，再配合工程師利用專業知識、工程經驗累積、和客戶方的需求設計。設計必須符合所有限制條件，即為一個可行設計。工程設計大多有多種或無線多種可行設計。在多種可行設計中利用有效的搜尋方法，尋找一個最佳或最適合的設計方案，就是最佳化設計。

桁架最佳化設計早在西元 1904 年由 Michell 等人[11]，文獻中提出桁

架結構最佳化設計，一般結構最佳化設計分為三類，

最佳桿件斷面尺寸設計(Size optimization)：

主要改變結構中桿件的斷面大小，並且符合相關規範和其他規定，稱為最佳桿件斷面尺寸設計。

最佳結構配置(Configuration optimization)：

主要改變結構中節點位置。

最佳拓樸設計(Topology optimization)：

主要改變結構中材料分配、節點、和桿件數目。

而最佳桿件斷面尺寸設計於 1974 年 Schmit[12]提出成為三種最佳化設計方法中最早發展的。

三種結構最佳化方法中，以最佳桿件斷面尺寸設計最為普遍，此方法針對已知結構物桿件配置。在符合應力限制、位移限制、和變形限制下，改變桿件斷面尺寸，達到最小總重量或最低成本等設計目標。本研究專注於最小總重量，桿件所受應力應限制於最大材料容許應力內，各自由度位移限制在最大位移限制內。

假設一桁架結構物由s個桿件所組成，自由度為t個則目標函數和限制函數分別表示為(式 3)、(式 4)、和(式 5)。

目標函數：

$$\min_{CSm<A_i<CSM} f(A) = \sum_{i=1}^{s} \rho_i A_i L_i \tag{3}$$

限制函數：

$$\sigma_i \leq \sigma_{i,allow}\text{，}\quad i = 1, 2, \ldots, s \tag{4}$$

$$\Delta_j \leq \Delta_{j,allow}\text{，}\quad j = 1, 2, \ldots, t \tag{5}$$

其中：

CSm：設計斷面下界

CSM：設計斷面上界

A_i：第i根桿件的斷面積

ρ_i：第i根桿件的密度

L_i：第i根桿件的長度

σ_i：第i根桿件所受應力

$\sigma_{i,allow}$：第i根桿件最大容許應力

Δ_j：第j個自由度的位移

$\Delta_{j,allow}$：第j個自由度的最大容許位移

3.3.2 桁架結構分析

桁架結構分析採用直接進度法分析桁架結構整體勁度、節點位移、桿

件應力、和支承反力，分析步驟如下

由材料力學可知桿件內力與節點位移關係

$$Q = ku \tag{6}$$

由圖 17，將(式 6)展開成

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} = \frac{EA}{L}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{7}$$

其中：

$Q_1 - Q_4$：局部座標中的內力

$u_1 - u_4$：局被座標中的變形

E：桿件材料的楊氏係數

A：桿件斷面積

L：桿件長度

由圖 17，利用平面座標轉換概念，局部座標和整體座標互相轉換關係如(式 8)和(式 9)所示

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \\ 0 & 0 & -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta \\ 0 & 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} \tag{9}$$

其中：

$F_1 - F_4$：整體座標中的內力

如(式 8)所示我們定義座標轉換矩陣T

$$T = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \\ 0 & 0 & -\sin\theta & \cos\theta \end{bmatrix} \tag{10}$$

(式 8) (式 9)可知

$$T^T = T^{-1} \tag{11}$$

因此局部座標和整體座標中位移轉換關係

$$u = Tv \tag{12}$$

將(式 12)帶入(式 6)得

$$Q = kTv \tag{13}$$

由(式 8)和(式 10)得

$$Q = TF \tag{14}$$

由(式 13)$Q = TF$ (14)(式 14)得

$$kTv = TF \tag{15}$$

(式 15)左右式互換且同乘T^{-1}，再由(式 11)得

$$F = T^{-1}kTv = T^{T}kTv = Kv \tag{16}$$

$$K = T^{T}kT = \frac{EA}{L}\begin{bmatrix} \cos^2\theta & \sin\theta\cos\theta & -\cos^2\theta & -\sin\theta\cos\theta \\ \sin\theta\cos\theta & \sin^2\theta & -\sin\theta\cos\theta & -\sin^2\theta \\ -\cos^2\theta & -\sin\theta\cos\theta & \cos^2\theta & \sin\theta\cos\theta \\ -\sin\theta\cos\theta & -\sin^2\theta & \sin\theta\cos\theta & \sin^2\theta \end{bmatrix} \tag{17}$$

其中K為一支桿件在整體座標中的勁度矩陣，將每支桿件K依相同自由度編號相加，即可得到桁架結構整體座標中的勁度矩陣S

$$S = \begin{bmatrix} K_{1\,1} & \cdots & K_{1\,t} \\ \vdots & \ddots & \vdots \\ K_{t\,1} & \cdots & K_{t\,t} \end{bmatrix}_{t\times t} \tag{18}$$

其中:

S：完整結構在整體座標下之勁度矩陣

t：自由度總數

將節點受力P和位移 d 配合矩陣S自由度順序排列

$$P = Sd \tag{19}$$

$$P = \begin{bmatrix} P_1 \\ \vdots \\ P_t \end{bmatrix} \tag{20}$$

$$d = \begin{bmatrix} d_1 \\ \vdots \\ d_t \end{bmatrix} \tag{21}$$

其中：

P：所有節點受力矩陣

d：所有節點位移矩陣

t：自由度總數

將(式 19)分為

$$\begin{bmatrix} P_f \\ P_s \end{bmatrix} = \begin{bmatrix} S_{f\,f} & S_{f\,s} \\ S_{s\,f} & S_{s\,s} \end{bmatrix} \begin{bmatrix} d_f \\ d_s \end{bmatrix} \tag{22}$$

其中：

f：外力點

s：反力點

將(式 22)利用矩陣乘法展開

$$P_f = S_{f\,f} d_f + S_{f\,s} d_s \tag{23}$$

$$P_s = S_{s\,f} d_f + S_{s\,s} d_s \tag{24}$$

已知反力點為支承，理論上並無位移(d_s為零矩陣)，因此(式 23)和(式 24)可簡化為

$$P_f = S_{f\,f} d_f \tag{25}$$

$$P_s = S_{s\,f} d_f \tag{26}$$

將(式 25)左右式同乘${S_{f\,f}}^{-1}$可得節點位移

$$d_f = {S_{f\,f}}^{-1} P_f \tag{27}$$

利用節點位移再將(式 16)代入(式 14)可得桿件內力

$$Q = TKv \tag{28}$$

求得桿件內力後透過已知斷面積求得桿件應力

$$\sigma = \frac{Q}{A} \tag{29}$$

利用(式 27)和(式 29)得到節點位移和桿件應力，即可與最大容許位移和最大容許應力做比較，檢查該結構物是否符合設計限制條件。

3.4 加速比

在平行運算領域中，加速比(Speedup Factor)用於表示當平行演算法與相對應的循序演算法，互相比較下速度快多少。

程式加速比：

$$S(p) = \frac{T_S}{T_p} \tag{30}$$

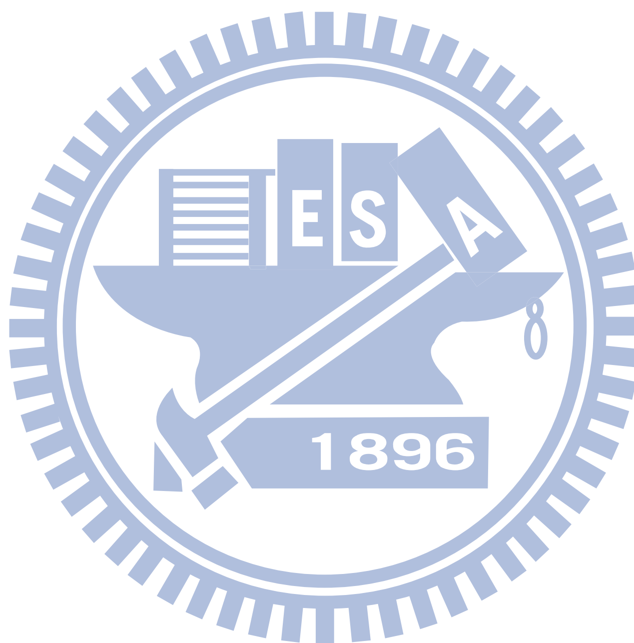
其中：

T_S：**最好**的循序演算法的執行時間

T_p：使用p個處理器的執行時間

效能提升有三種情形如圖 19 分為理論值、一般情形、和超線性。理論值代表加速比與核心成斜率為 1 的情形，也就是說假設 4 核心的 CPU 的加速比理論值就是 4。超線性只有在某些情形會發生，例如當程式有特

別為平行運算最佳化或有額外的硬體支援例如記憶體或快取。本研究因無【最好的循序演算法】作為T_S。因此定義T_S為單執行緒執行所耗時間，來計算加速比。



第四章 結果分析與探討

本研究分別利用矩陣乘法和粒子群最佳化分別利用『OpenMP』平行運算做為測試範例。其中粒子群最佳化包括【30 個變數平方和之最小值】它是 benchmark 題目之一、【Beale's function】和【桁架斷面結構最佳化】，以上測試環境皆是在同一個平台下，詳細測試環境如下：

中央處理器：intel i5-2400 (6M Cache, up to 3.40 GHz) 四核心

記憶體：8 GB

作業系統：Windows 7 ultimate, 64bit

開發平台：Visual Studio 2012

4.1 平行運算矩陣乘法

編寫循序矩陣乘法如圖 11 即可，編寫開始先建立三個相同大小矩陣空間分別為 ArrA, ArrB, ArrC, 和 ArrD。將 ArrA 和 ArrB 的每個元素值以隨機亂數產生後，兩矩陣相乘存入 ArrD。平行矩陣乘法如圖 12 詳細編寫過程已於 3.1 節介紹，相乘後存入 ArrC。將兩矩陣 ArrC 和 ArrD 相減驗證是否正確。測試矩陣列行值分別為25×25、50×50、100×100、200×200、400×400、800×800、和1600×1600，為了

避免矩陣太小而無法量測其計算時間，且讓測試時間結果更加準確無論循序程式或平行程式都重複計算 100 次。執行結果如表 3，表中顯示計算時間隨著矩陣大小增加且計算上皆無誤差。而效能如圖 18，由圖可知當矩陣大小在25×25平行效果並不好，表 3 的加速因子也顯示矩陣大小在25×25為 0.07483 這告訴我們這個矩陣大小並不適合平行。主要原因在於程式中【Fork】和【Join】都會消耗一些效能，當矩陣太小時平行反而執行時間大於循序執行。當矩陣大小從50×50到200×200時效能開始上升，加速因子一路攀升至 3.71，平行計算已經比循序計算增加 2.71 倍。當矩陣大小到200×200後上升幅度因硬體限制之下逐漸趨緩，矩陣大小到1600×1600加速因子為 3.83 已經與先前加速因此理論值 4 接近。因此在矩陣小於1600×1600時越大的矩陣有越好的平行效果。

4.2 平行運算粒子群最佳化

由圖 16 可知粒子群最佳化的循序演算法，在 3.2 節提到本研究將在迭代搜尋時(do-while)迴圈內做平行。研究初期本以為在圖 16 第 13 行處 for 迴圈掛上『OpenMP』的 Compiler Directive 指令(#pragma omp for)就可以將原循序程式成功平行化。程式的確以多執行緒執行但平行效果並不好，造成平行效果不佳原因及解決方案在 4.2.1 測試結果說明。

4.2.1 30 個變數平方和之最小值

由圖 16 可知粒子群最佳化的循序演算法。第 13 行是 for 迴圈，迴圈次數為粒子總數，又先前 3.1 節提過，將 for 迴圈前增加一列『OpenMP』的 Compiler Directive 指令(#pragma omp for)平行化。雖然效能有明顯上升，但是仔細觀察其最佳化結果在相同粒子數和迭代次數下，此平行演算法明顯比循序演算法明顯較差。原因在於多執行緒同時和同一記憶體位置作存取造成競爭危害(race condition)。利用一種情形說明，假設圖 20 是一計算目標函數所需資料，而多個執行緒同時需要存取此記憶體空間就可造成競爭危害。因此本研究將須平行運算的資料全部複製三份一共四份，如此一來就可以讓四個執行緒同時存取達到平行的效果。利用『OpenMP』的 Compiler Directive 中 section 的功能將粒子數分為四份，每個執行緒分別計算四分之一的粒子數。此次測試迭代次數皆為十萬次，c_1和c_2皆為 2，修改演算法如圖 21 和圖 22，結果如表 4 和圖 23。可知粒子群最佳化非常適合平行化運算，僅僅使用 4 個粒子作搜尋加速比還有 1.65，當粒子數增加到 10,000 加速比 3.69，當粒子數再往上加到 200,000 加速比為 3.73 效能達到最高到。本研究發現粒子群最佳化中圖 21 第 40 行也就是尋找族群最佳位置時這部分本為循序執行。當總粒子數越大此循序執行越久，因

此嘗試將此迴圈也分為四份分別放入 section 中。讓每一個 section 中先搜尋自己的最佳解，離開 section 後再由四個最佳解中挑選一個最好的。修改後發現效果不如預期中好如表 6，反而在粒子數比較少時因迴圈數太小無法顯示平行效果。當粒子數達到一百萬時加速比比修改前增加 0.04。此測試案例設定迭代停止條件為目標函數值小於 1 時收斂情形如圖 24，最佳解和目標函數值如表 5。

4.2.2 Beale's function

此測試案例也是最佳化測試題目之一如同 4.2.1，Beale's function 如下

$$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

最佳解落在$f(3,0.5) = 0$，搜尋範圍$-4.5 \leq x,y \leq 4.5$，迭代次數皆為二十萬次。方程式繪製如圖 25。測試效能如表 7 和圖 26，收斂過程如圖 27。觀察到因此案例維度較小因此加速比上升有限，當粒子數 10000 時加速比為 3.3。由方程式圖 25 可知此案例很容易收斂至最佳解，無論粒子數多寡幾乎都能目標函數值收斂到 0 或非常接近 0。

4.2.3 桁架斷面結構最佳化

在改寫成平行化運算粒子群最佳化桁架結構遇到一些瓶頸。其中包括資料衝突，多個執行緒同時需要讀取計算目標函數所需資料，例如桿

件編號、節點編號、材料參數等，造成執行緒排隊讀取資料。當這些計算所需資料還沒複製出多個副本前，無論如何調整粒子數平行化的效能都比循序執行來得低。最難除錯在於斷面參數陣列沒有複製出多個副本，原因在於因為斷面參數是核心程式的變數。平行運算下效能有上升，但收斂速度明顯慢得多，甚至不收斂。再循序計算下又可以收斂的情形下實在難以除錯。最後逐行檢查下才發現這個重大錯誤。這也是編寫平行程式最難的地方。

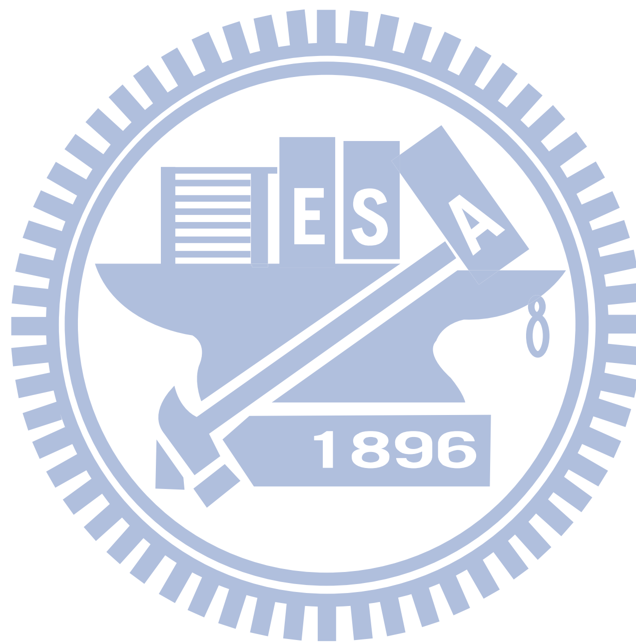
本研究測試案例如圖 28 是一 10 支桿件的桁架結構。此案例先前已有許多學者使用 Li 等人[13]、Perez 等人[14]、Kaveh 等人[15, 16]。因此利用此案例來驗證程式是否準確。此案例材料密度$\rho = 0.1\, {}^{lb}/_{in^3}$、彈性係數$E = 10000\, ksi$，桁架一共 10 支桿件 6 個節點。節點和桿件標號如圖 28。每一隻桿件都有個別的設計斷面，斷面設計限制於$0.1\, in^2$到 $35.0\, in^2$，應力設計限制$\pm 25.0\, ksi$，節點水平或垂直位移限制$\pm 2.0\, in$。受力$P_1 = 100\, kips$和$P_2 = 0\, kips$。粒子群最佳化參數c_1和c_2皆為 2，迭代次數皆為一萬次。斷面最佳結果如表 8，和文獻 Li 等人[13]所使用的粒子群最佳化比較結果還好。原因可能在於本研究利用多粒子數和大量的迭代次數搜尋以相同的演算法得到更好的解。平行效果如表 9 與圖 29，可以觀察到與 4.2.1 節的測試案例相差不遠。隨著粒子數增加效能

也逐漸提高。值得注意的是效能在此試驗因為目標函數改為結構分析多了許多載入目標函數的資料造成平行效果稍微降低，當粒子數為 5000 時加速比為 3.43。第二次測試利用迭代次數觀察對效能的影響，粒子群最佳化設定參數c_1和c_2皆為 2，粒子數固定為 5000。結果如表 10 當迭代次數增加效能也會隨著增加，原因是每次迭代平行運算部分都會比循序運算節省時間。因此當迭代次數小於 10000 加速比隨著迭代次數增加而增加。桁架結構分析結果如表 11 和表 12 皆在限制範圍內，而收斂過程如圖 30。

第二個桁架測試案例如圖 31，17 支桁架結構。利用相同方式作桁架斷面最佳化設計，此案例材料密度$\rho = 0.268\ {}^{lb}/_{in^3}$、彈性係數$E = 30000\ ksi$，桁架一共 17 支桿件 9 個節點，應力限制$\pm 50\ ksi$，位移限制$\pm 2\ in$。效能測試結果如表 14 和圖 32，此案例是 17 支桁架結構比上個案例多了 7 個維度平行化計算量更大，因此加速比提升至 3.54。收斂情形如圖 33，最佳化斷面結果如表 13。迭代次數多在相同演算法下較 Li 等人[13]好，結構分析如表 15 和表 16，皆符合再限制條件內。

第三個桁架測試案例如圖 34，26 支桁架結構，方法同上作桁架斷面最佳化設計。此案例材料密度$\rho = 7.27 \times 10^{-8}\ {}^{KN}/_{mm^3}$、彈性係數$E = 207\ GPa$，桁架一共 26 支桿件 15 個節點，桿件編號如表 17，應力

限制±0.334 GPa，位移限制±50.8 mm。效能測試結果如表 18 和圖 35，收斂情形如圖 36，最佳化斷面後結構分析桿件應力和位移如表 19 和表 20。由此案例可觀察應桿件數更多而維度增加，讓平行部分計算量更多，當粒子總數 4 個加速比已逼近 3，當粒子數 5000 最高加速比達到 3.64。



第五章 結論與建議

平行運算粒子群最佳化程式已經開發完成。充分使用現今 CPU 的多核心架構，而達到較好的運算效能和減少運算時間。利用多種目標函數測試驗證本演算法可行，因此可得以下結論。

5.1 結論

1. 本研究明顯看出電腦硬體不斷在進步和變化，CPU 從原本的單核心演變至多核心，編寫軟體時也可以多注意是否可以平行化處理。雖然比起循序程式會多花時間在除錯，一旦成功程式效能就會隨著硬體規格上升。
2. 在平行矩陣乘法的結果中明顯看出加速比 3.83 因為整個矩陣乘法是可以平行的演算法，因此在效能有非常顯著的提升。
3. 【30 個變數平方和之最小值】利用粒子群最佳化去求解，因為粒子群最佳化迭代中在尋找族群最佳解無法完全平行，因此效能比矩陣乘法低一些。
4. 【桁架斷面結構最佳化】因為結構最佳化目標函數是結構分析，因此比 4.2.1 節需要載入更多資料，效能當然也會稍微減損，不過當粒子數

5000 和迭代次數 10000 以上時加速比還是維持在 3.2 以上。17 支桁架和 26 支桁架結構下因粒子群最佳化維度增加，加速比些許上升。

5. 此平行粒子群最佳化程式只需更改目標函數即可做更多的應用，並不局限於【30 個變數平方和之最小值】和【桁架斷面結構最佳化】兩種案例。

5.2 建議

1. 多核心處理器是目前電腦主流使用的，但近年新崛起的 GPGPU(General-purpose computing on graphics processing units)，也就是利用電腦中顯示卡來做運算。利用顯示卡大量的核心做更龐大的運算，但平行化的困難度比使用多核心 CPU 平行還難得許多，顯示卡記憶體空間也比電腦記憶體來的小，因此控管顯示卡記憶體是編寫 GPGPU 程式一大學問。
2. 將來程式可以改用 Object-oriented programming (OOP)的方式編寫，可以讓程式有更好的可讀性，方便未來配合使用者或硬體做修改或維護。例如讓粒子群最佳化中粒子設為一個物件，物件中有$X_{i,j}^k$、$P_{i,j}^k$、和$V_{i,j}^k$等參數。

參考文獻

[1] "High Performance Computing Training," *https://computing.llnl.gov/?set=training&page=index*.

[2] 郭信川，官佳慶, "隨機搜尋法於多極值最佳化問題之應用," *中國造船暨輪機工程學刊,* vol. 19, pp. 33-40, 民國八十九年.

[3] 鄭博育, "啟發式桁架斷面尺寸最佳化設計," *國立交通大學,* vol. 碩士論文, 2005.

[4] *平行計算基礎理論,系統及應用研究:* 國立中正大學資訊工程研究所, 1992.

[5] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM,* vol. 31, pp. 532-533, 1988.

[6] L. L. N. L. Blaise Barney, "OpenMP."

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 1995, pp. 1942-1948.

[8] J. F. Kennedy, J. Kennedy, and R. C. Eberhart, *Swarm intelligence*: Morgan Kaufmann Pub, 2001.

[9] Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, 2001, pp. 81-86.

[10] 蘇維農, "JMC: 基於 OpenMP 3.0 平行程式設計模型, 實作支援 OpenMP 的 Java 語言編譯器及執行期間程式庫," *清華大學資訊工程學系學位論文,* 2010.

[11] A. G. M. Michell, "LVIII. The limits of economy of material in frame-

structures," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science,* vol. 8, pp. 589-597, 1904.

[12] L. A. Schmit and B. Farshi, "Some approximation concepts for structural synthesis," *AIAA journal,* vol. 12, pp. 692-699, 1974.

[13] L. Li, Z. Huang, F. Liu, and Q. Wu, "A heuristic particle swarm optimizer for optimization of pin connected structures," *Computers & Structures,* vol. 85, pp. 340-349, 2007.

[14] R. Perez and K. Behdinan, "Particle swarm approach for structural design optimization," *Computers & Structures,* vol. 85, pp. 1579-1588, 2007.

[15] A. Kaveh and S. Talatahari, "A hybrid particle swarm and ant colony optimization for design of truss structures," *Asian J Civil Eng,* vol. 9, pp. 329-48, 2008.

[16] A. Kaveh and S. Talatahari, "Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures," *Computers & Structures,* vol. 87, pp. 267-283, 2009.

[17] Flynn, Michael J. "*Some computer organizations and their effectiveness.*"Computers, IEEE Transactions on 100.9 (1972): 948-960.

[18] 張舒、祩絕利、趙開勇, "*GPU 高性能運算之 CUDA*," *松崗出版社,* 2011.

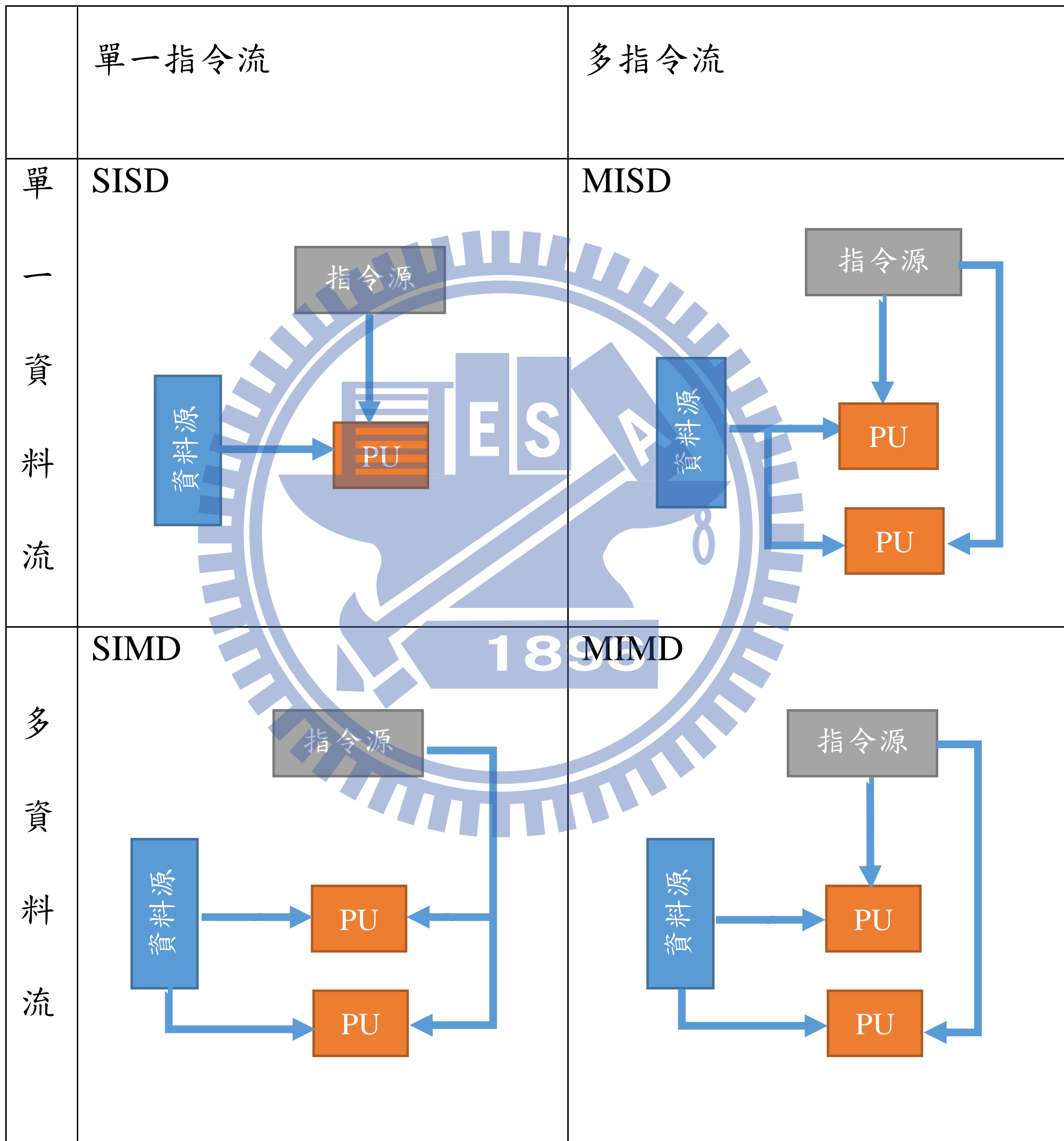
[19] Erbatur, Fuat, et al. "*Optimal design of planar and space structures with genetic algorithms.*" Computers & Structures 75.2 (2000): 209-224.

[20] Aarts, Emile, and Jan Korst. "*Simulated annealing and Boltzmann machines.*" (1988).

[21] Glover, F., & Laguna, M. (1997). *Tabu search* (Vol. 22). Boston: Kluwer academic publishers.

附表

表 1　費林分類法[17]



	單一指令流	多指令流
單一資料流	SISD	MISD
多資料流	SIMD	MIMD

表 2　OpenMP Directives

	directive-name	[clause, ...]	
#pragma omp	parallel do for	非必要 例如：shared(x)	(Enter)

表 3　矩陣乘法測試結果

矩陣列行大小(m*n)	**循序（秒)**	**平行(秒)**	**加速比**	**誤差**
25	0.011	0.147	0.07483	0.0
50	0.056	0.037	1.513514	0.0
100	0.462	0.16	2.817073	0.0
200	3.711	1.169	3.174508	0.0
400	35	10.783	3.24585	0.0
800	396.67	115.92	3.421929	0.0
1600	4669.6	1216.67	3.838017	0.0

表 4　30 個變數的最小平方和最佳化執行時間與加速比

粒子總數	循序(秒)	平行(秒)	加速比
4	1.923	1.163	1.653482
20	8.647	3.329	2.597477
60	25.535	8.358	3.055157
120	51.466	15.459	3.329193
240	102.49	29.807	3.438454
500	209.616	63.54	3.298961
1000	422.97	125.87	3.360372
10000	4231.897	1150.45	3.678471
100000	41819.5	11395.7	3.669761
200000	83466.2	22381.6	3.729233
500000	207222	55882.2	3.708193

表 5　30 個變數的最小平方和最佳化執行結果

變數	變數值
1	0.0889
2	0.0657
3	-0.0166
4	0.0381
5	0.0095
6	0.2
7	-0.1084
8	-0.0029
9	0.1548
10	-0.1425
11	-0.0601
12	0.1514
13	-0.3342
14	-0.0039
15	-0.09
16	-0.1522
17	0.1766
18	0.1782
19	-0.0439
20	0.4742
21	0.1943
22	-0.1191
23	-0.3156
24	-0.0576
25	-0.1914
26	-0.251
27	-0.0625
28	0.1707
29	0.0273
30	0.1035
目標函數值：0.868313611	

表 6　修改平行程式後測試

粒子總數	修改前(秒)	修改後(秒)	相差(秒)	加速比差異
4	1.163	1.495	-0.332	-0.36719
20	3.329	3.44	-0.11	-0.08016
60	8.358	8.97	-0.612	-0.20845
120	15.459	15.712	-0.253	-0.05361
240	29.807	31.06	-1.253	-0.13871
500	63.54	64.133	-0.593	-0.0305
1000	125.87	125.236	0.634	0.017012
10000	1150.45	1147.19	3.26	0.010453
100000	11395.7	11358.4	37.3	0.012051
200000	22381.6	22336.2	45.399	0.00758
500000	55882.2	55488	394.2	0.026344
1000000	110270	109083	11870	0.0402

表 7　Beale's function 粒子數與加速比關係

粒子總數	循序(秒)	平行(秒)	加速比
20	1.884	1.31	1.43
60	5.023	2.525	1.99
120	9.879	4.105	2.41
240	19.61	7.073	2.77
500	39.998	13.691	2.92
1000	79.639	27.92	2.85
5000	393.07	125.341	3.14
10000	782.09	237.309	3.30

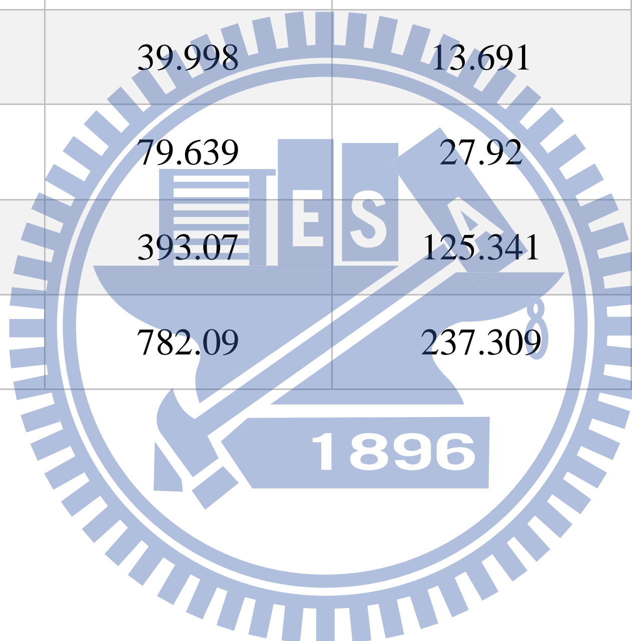


表 8　10 支桿件桁架斷面最佳化結果

斷面編號	PSO (in^2)	PSO (Li[13]) (in^2)
A_1	31.50804	33.469
A_2	0.152502	0.110
A_3	25.29146	23.177
A_4	14.25196	15.475
A_5	0.11103	3.649
A_6	0.326716	0.116
A_7	7.062967	8.328
A_8	18.94086	23.34
A_9	24.01386	23.014
A_{10}	0.108947	0.190
總重(lb)	**5131**	**5529.50**

表 9　10 支桿件最佳化執行時間與粒子總數關係

粒子總數	循序(秒)	平行(秒)	加速比
4	0.897	0.391	2.29
20	4.195	1.605	2.61
40	8.464	3.06	2.76
100	24.328	8.415	2.89
500	103.532	32.412	3.19
1000	207.76	63.445	3.27
5000	1017.89	296.806	3.42

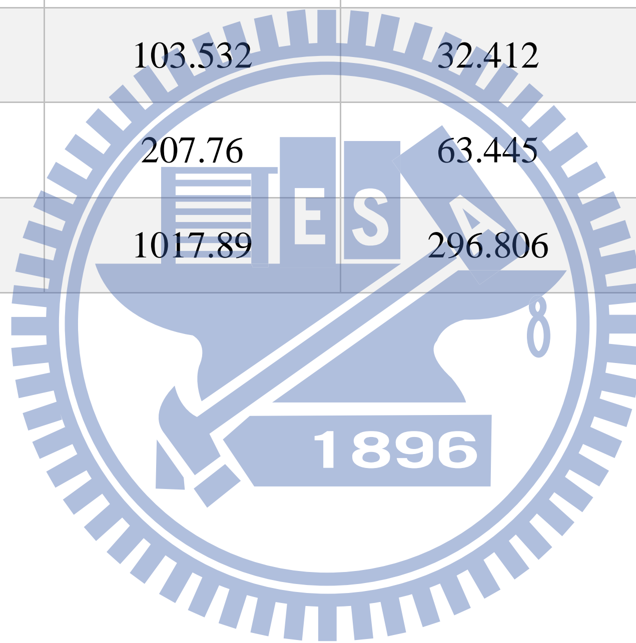


表 10　10 支桿件最佳化和迭代次數關係

迭代次數	循序執行(秒)	平行執行(秒)	加速比
20	2.199	1.491	1.47
50	5.176	2.502	2.07
100	10.601	4.477	2.37
500	52.533	16.983	3.09
1000	103.133	32.499	3.17
5000	513.57	153.528	3.35
10000	1017.89	296.806	3.43
50000	5132.183	1498.93	3.42

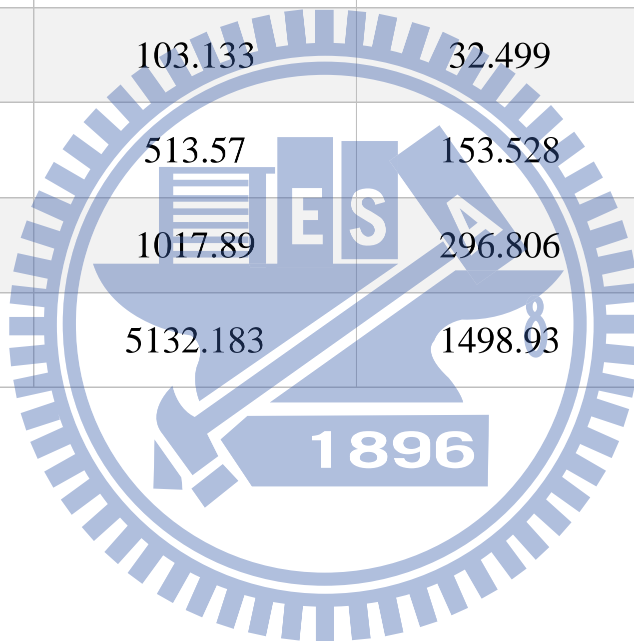


表 11　10 支桿件最佳化斷面後節點位移

節點編號	X 方向 (in)	Y 方向 (in)
1	0.488	-1.75
2	-0.378	-1.80
3	0.423	-0.741
4	-0.296	-0.775
5	0	0
6	0	0

註：

節點 5 和 6 為鉸支承

限制為 ±2 in

表 12　10 支桿件最佳化斷面後結構分析桿件應力

桿件編號	桿件應力 (ksi)
1	-7.29
2	0.81
3	8.00
4	7.03
5	-22.05
6	0.21
7	-17.82
8	7.15
9	-5.73
10	-1.27

註：

限制為 ±25 ksi

表 13　17 支桿件桁架斷面最佳化結果

斷面(in^2)	PSO	PSO (Li[13])
A_1	14.024	15.766
A_2	1.200	2.263
A_3	13.217	13.854
A_4	0.222	0.106
A_5	10.037	11.356
A_6	4.448	3.915
A_7	10.025	8.071
A_8	1.536	0.100
A_9	8.397	5.850
A_{10}	0.160	2.294
A_{11}	4.412	6.313
A_{12}	0.185	3.375
A_{13}	5.339	5.434
A_{14}	5.373	3.918
A_{15}	3.978	3.534
A_{16}	2.158	2.314
A_{17}	5.537	3.542
總重(lb)	**2670.23**	**2724.37**

表 14　17 支桿件最佳化和粒子總數關係

粒子總數	循序執行(秒)	平行執行(秒)	加速比
4	0.948	0.465	2.04
20	4.297	2.092	2.05
40	8.78	3.94	2.23
100	21.127	9.005	2.35
500	106.939	35.042	3.05
1000	212.283	60.907	3.49
5000	1058.427	298.65	3.54

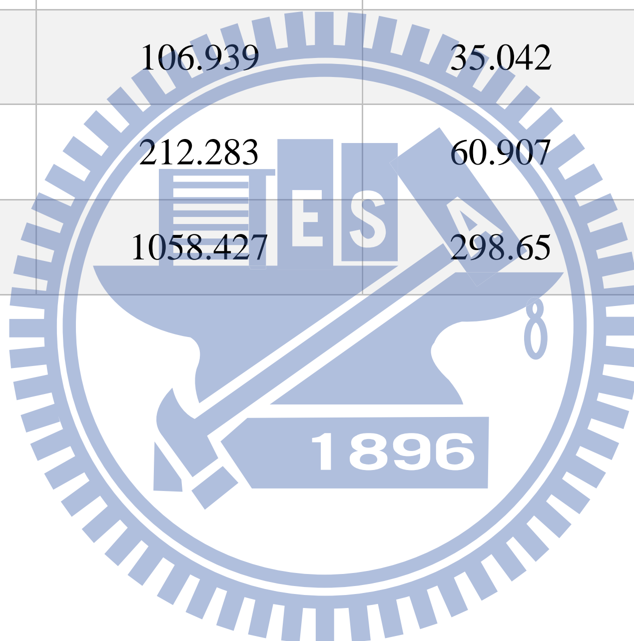


表 15　17 支桁架結構最佳化斷面後節點位移

節點編號	**X 方向 (in)**	**Y 方向 (in)**
1	0	0
2	0	0
3	-0.082	-0.278
4	0.089	-0.267
5	-0.173	-0.687
6	0.164	-0.637
7	-0.251	-1.217
8	0.242	-1.268
9	-0.313	-1.999

註：

節點 1 和 2 為鉸支承

限制為 ±2 in

表 16　17 支桁架結構最佳化斷面後結構分析桿件應力

桿件編號	桿件應力 (ksi)
1	-26.74
2	-29.45
3	24.59
4	-3.41
5	-22.49
6	-23.61
7	27.36
8	-14.92
9	-23.48
10	-24.79
11	23.30
12	15.17
13	-26.49
14	18.61
15	26.67
16	16.87
17	24.82

註：

限制為 ±50 ksi

表 17　26 支桁架結構桿件編號表

桿件編號	開始節點	結束節點
1	1	2
2	1	7
3	2	7
4	2	8
5	2	3
6	3	8
7	3	14
8	4	15
9	4	12
10	4	5
11	5	12
12	5	13
13	5	6
14	6	13
15	7	8
16	8	9
17	8	14
18	9	10
19	9	14
20	10	14
21	10	11
22	10	15
23	11	15
24	11	12
25	12	13
26	12	15

表 18　26 支桁架結構最佳化和粒子總數關係

粒子總數	循序執行(秒)	平行執行(秒)	加速比
4	2.344	0.795	2.95
20	11.506	3.666	3.14
40	22.945	7.151	3.21
100	56.192	17.553	3.20
500	281.287	84.064	3.35
1000	556.545	160.861	3.46
5000	2832.562	779.182	3.64

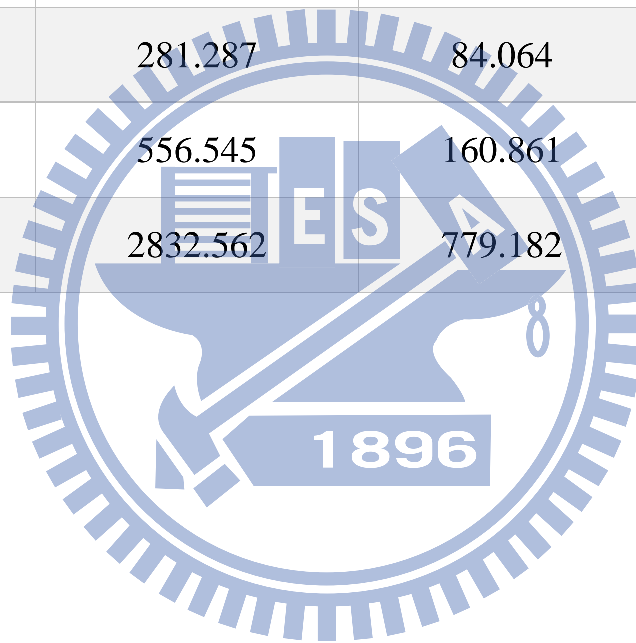


表 19　26 支桁架結構最佳化斷面後桿件應力

桿件編號	桿件應力 (GPa)
1	3.202E-02
2	-3.291E-02
3	2.012E-01
4	-1.873E-01
5	5.112E-02
6	3.371E-01
7	7.193E-02
8	7.193E-02
9	3.371E-01
10	5.112E-02
11	-1.873E-01
12	2.012E-01
13	3.202E-02
14	-3.291E-02
15	-4.836E-02
16	-4.577E-02
17	-1.647E-01
18	-6.333E-02
19	1.599E-01
20	5.936E-02
21	-6.333E-02
22	5.936E-02
23	1.599E-01
24	-4.577E-02
25	-4.836E-02
26	-1.647E-01

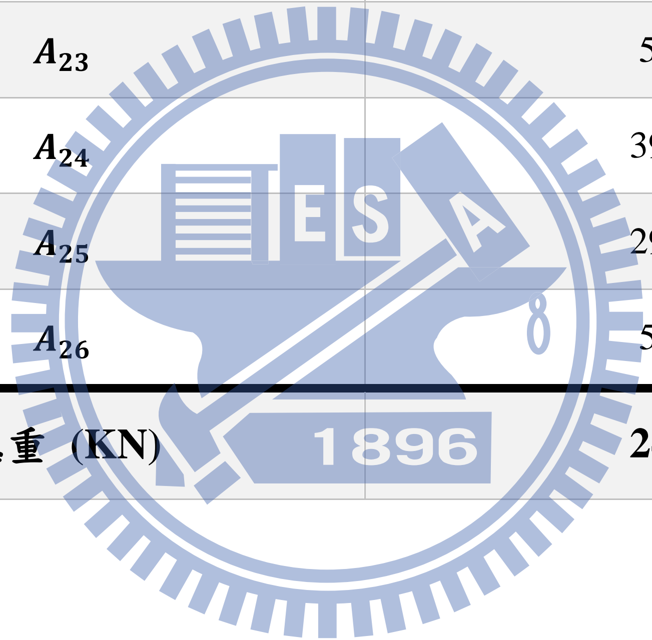
表 20　26 支桁架結構最佳化斷面後節點位移

節點編號	X 方向 (mm)	Y 方向 (mm)
1	5.94	-50.79
2	3.65	-45.15
3	0.00	0.00
4	0.00	0.00
5	-3.65	-45.15
6	-5.94	-50.79
7	7.98	-50.17
8	-1.29	-24.72
9	1.78	-24.33
10	0.00	-11.72
11	-1.78	-24.33
12	1.29	-24.72
13	-7.98	-50.17
14	10.47	-14.27
15	-10.47	-14.27

表 21　26 支桁架結構最佳化斷面結果

斷面編號	**斷面面積 (mm^2)**
A_1	2810
A_2	3057
A_3	400
A_4	481
A_5	3521
A_6	478
A_7	3754
A_8	3754
A_9	478
A_{10}	3521
A_{11}	481
A_{12}	400
A_{13}	2810
A_{14}	3057
A_{15}	2913
A_{16}	3957

A_{17}	547
A_{18}	3496
A_{19}	504
A_{20}	3033
A_{21}	3496
A_{22}	3033
A_{23}	504
A_{24}	3957
A_{25}	2913
A_{26}	547
總重 (KN)	**24.2**



附圖

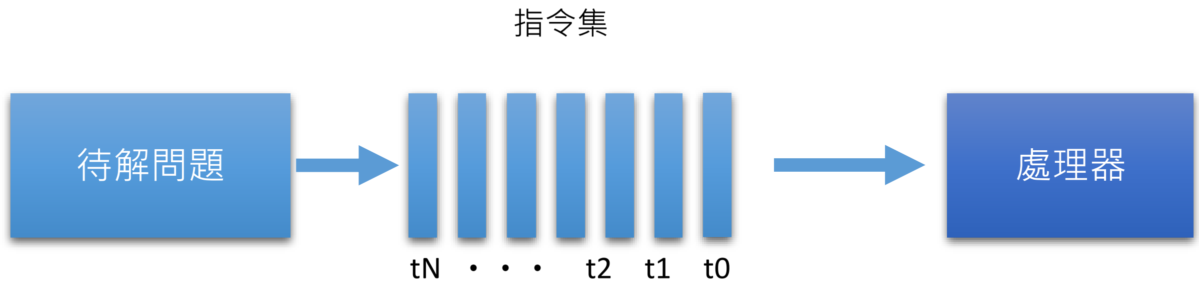


圖 1 傳統連續處理

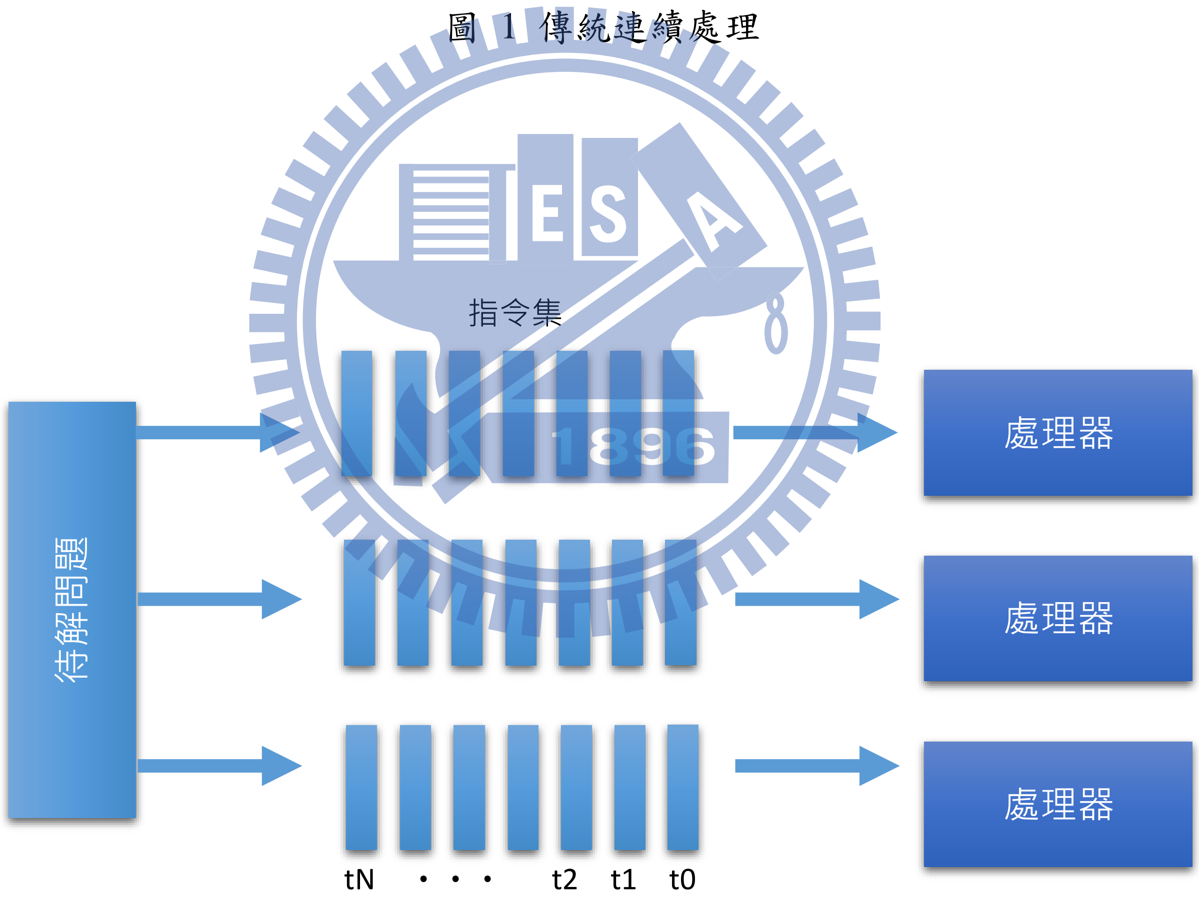


圖 2 平行處理

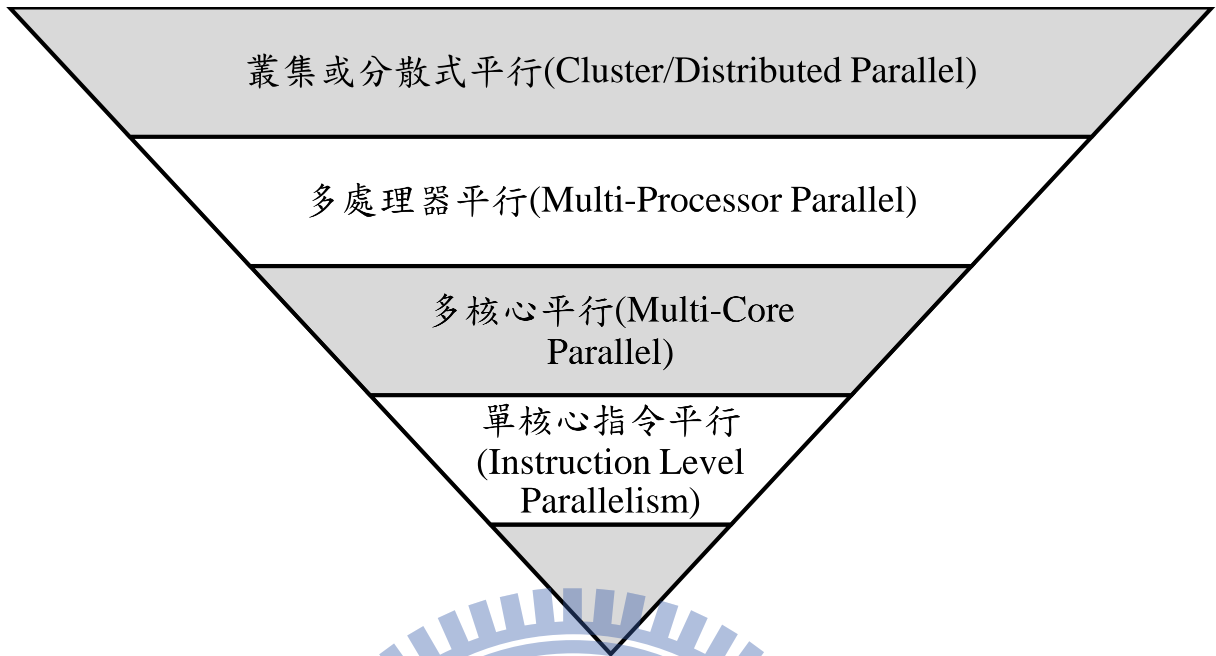


圖 3 平行層次圖[18]

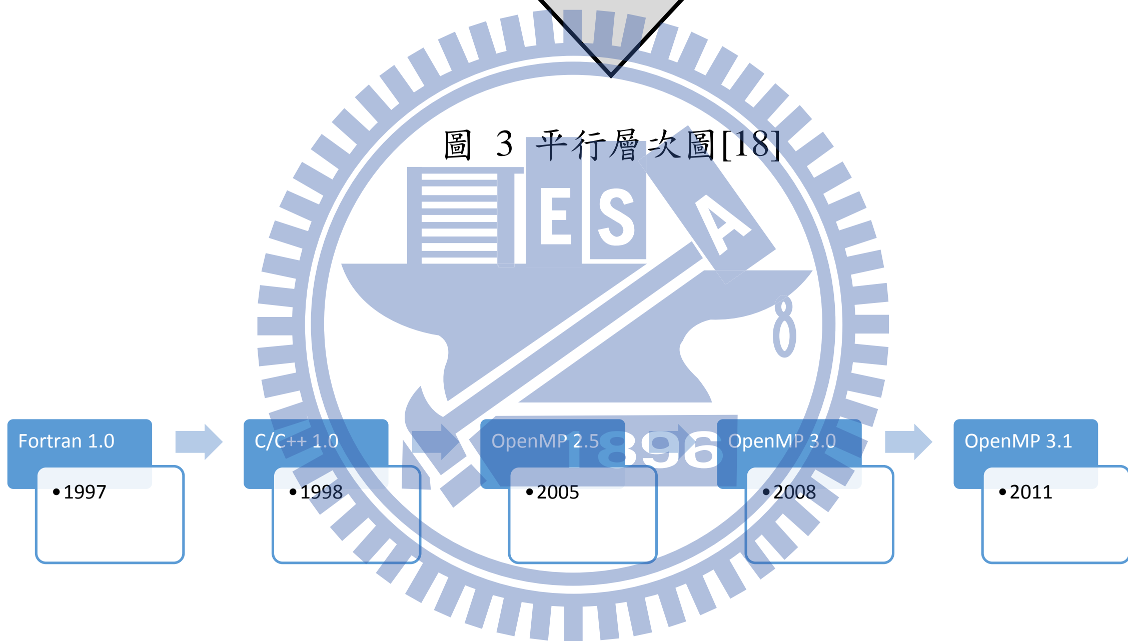


圖 4 OpenMP 發展流程圖

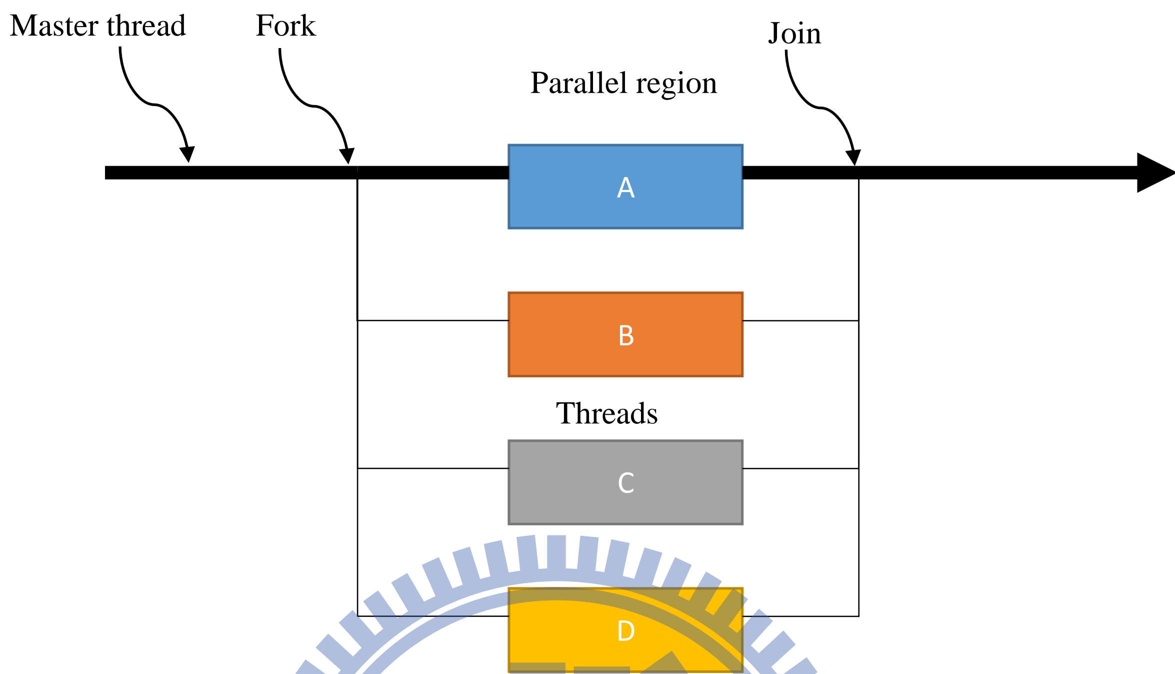


圖 5 Fork/Join Model

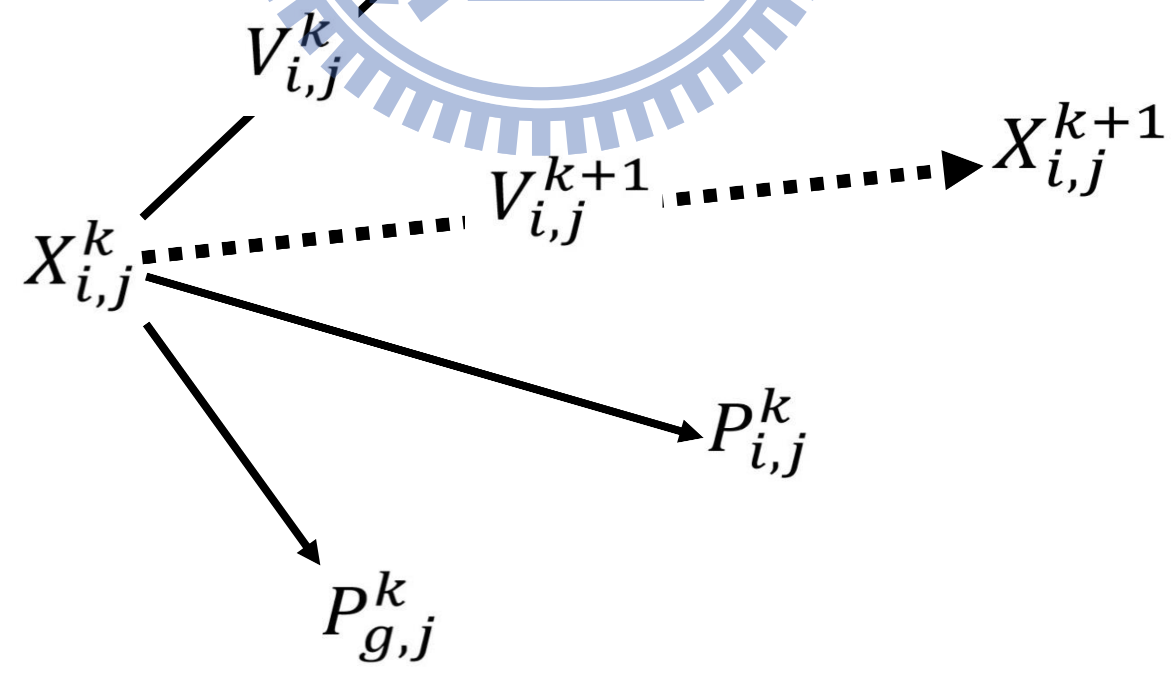


圖 6 粒子群最佳化粒子速度和位置更新示意圖

圖 7 粒子群最佳化流程圖

```
1  #include “omp.h”
2  void main()
3  {
4  #pragma omp parallel
5      {
6          int ID = omp_get_thread_num();
7          printf(“ hello(%d) ”, ID);
8          printf(“ world(%d) \n”, ID);
9      }
10 }
```

圖 8 簡易 OpenMP 程式

圖 9 簡易 OpenMP 程式執行結果

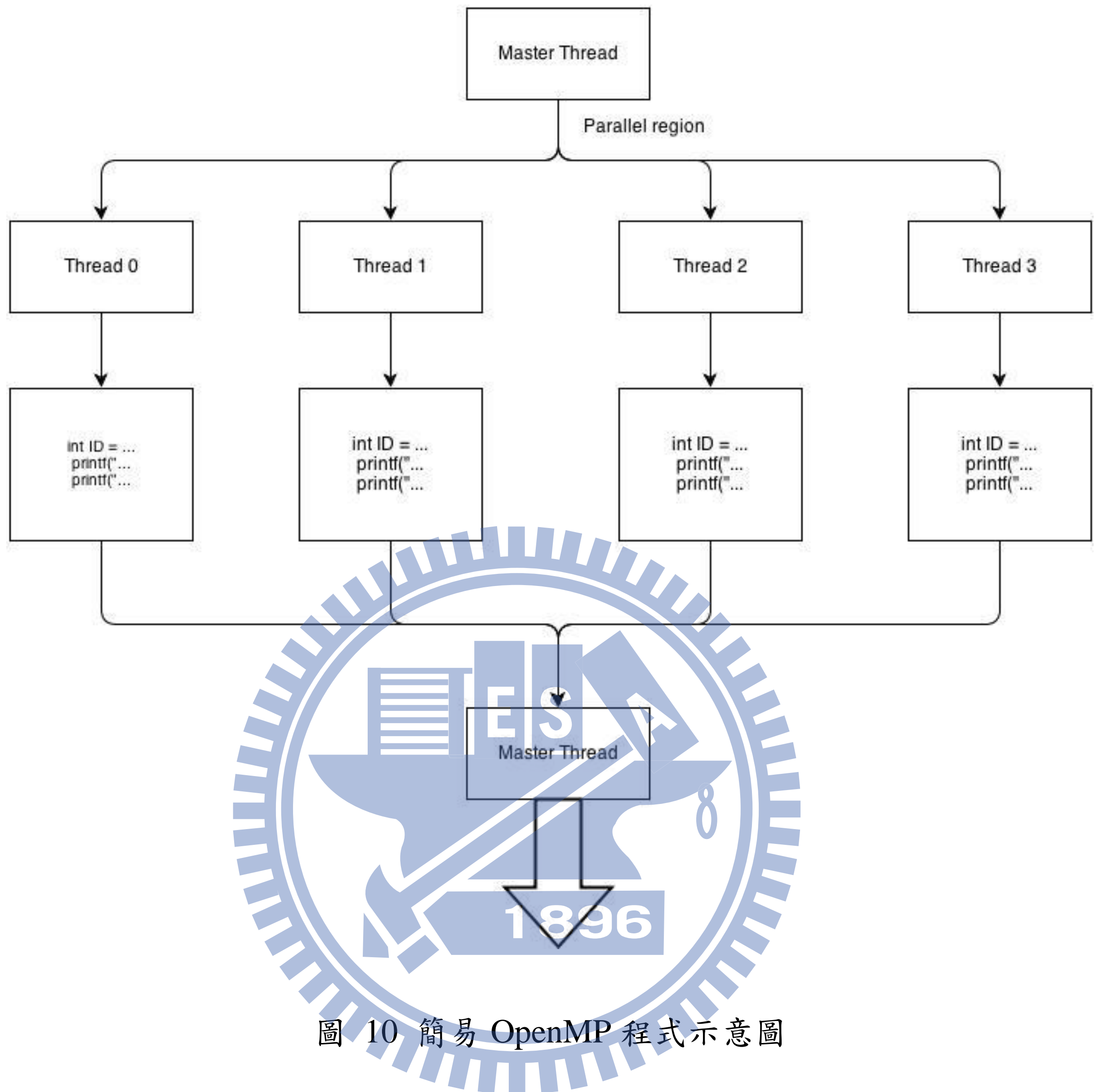


圖 10 簡易 OpenMP 程式示意圖

```
for (i=0; i<rowA; i++)
{
    for (j=0; j<colB; j++)
    {
        for (k=0; k<colA; k++)
        {
            ArrC[i][j] += ArrA[i][k]*ArrB[k][j];
        }
    }
}
```

圖 11 矩陣乘法

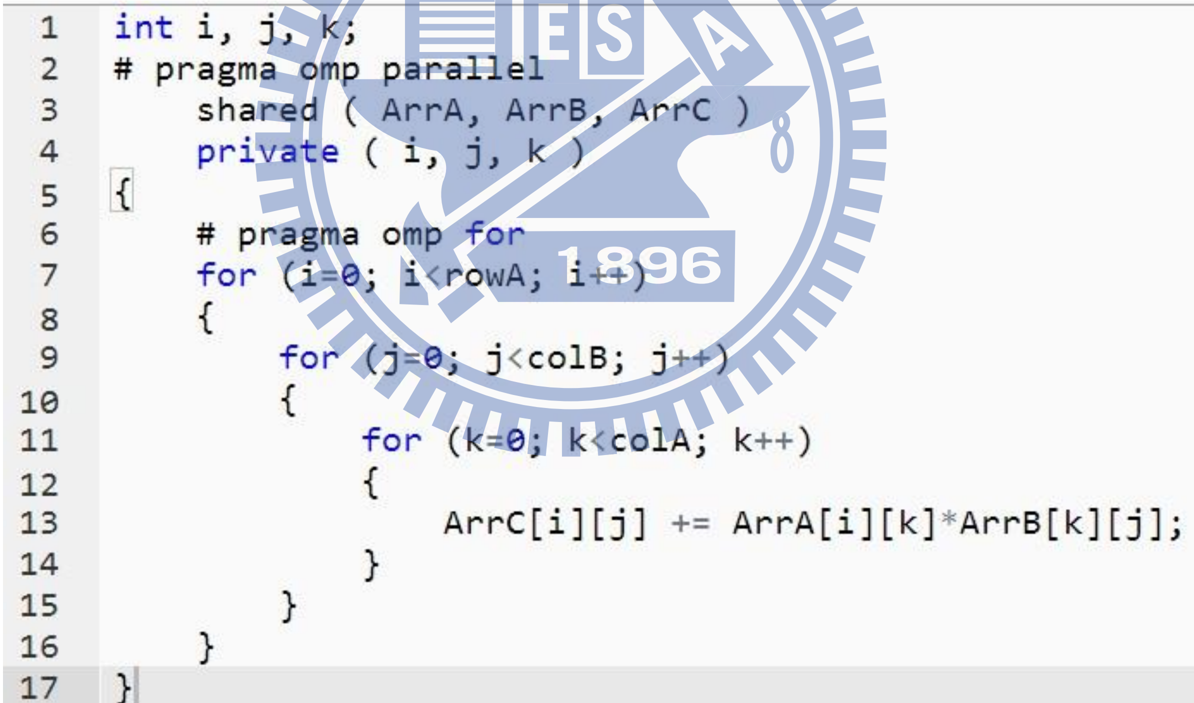


圖 12 矩陣乘法使用 OpenMP

```
1   int i, j, k=0;
2   #pragma omp parallel
3   {
4       #pragma omp for
5       for (i=0; i<4; i++){
6           for(j=0; j<4; j++){
7               printf(" %d + %d = %d by tid = %d\n",
8                   i, j, i+j, omp_get_thread_num());
9               k++;
10          }
11      }
12  }
13  printf(" 執行次數 = %d\n", k);
```

圖 13 競爭危害範例

圖 14 競爭危害輸出結果

```
int i, j, k=0;
#pragma omp parallel
{
    #pragma omp for private(j)
    for (i=0; i<4; i++){
        for(j=0; j<4; j++){
            printf(" %d + %d = %d by tid = %d\n",
                i, j, i+j, omp_get_thread_num());
            k++;
        }
    }
}
printf(" 執行次數 = %d\n", k);
```

圖 15 解決競爭危害

	Pseudocode
1：	for (i=0; i<粒子總數; i++)
2：	for (j=0; j<維度; j++)
3：	初始化：$X_{i,j}^{0}$、$V_{i,j}^{0}$、$P_{i,j}^{0}$
4：	計算目標函數值
5：	end for
6：	end for
7：	for (i=0; i<粒子總數; i++)
8：	if($f(P_g^0)>f(X_i^0)$)
9：	$P_{g,j}^{0}=X_{i,j}^{0}$;
10：	end for
11：	do{
12：	隨機產生r_1, r_2
13：	for (i=0; i<粒子總數; i++) {
14：	for (j=0; j<維度; j++) {
15：	$V_{i,j}^{k+1}=V_{i,j}^{k}+c_1r_1\left(P_{i,j}^{k}-X_{i,j}^{k}\right)+c_2r_2\left(P_{g,j}^{k}-X_{i,j}^{k}\right)$;
16：	$X_{i,j}^{k+1}=X_{i,j}^{k}+V_{i,j}^{k+1}$;
17：	end for
18：	計算目標函數值$f\left(X_{i,j}^{k+1}\right)$;
19：	if ($f\left(P_i^k\right)>f\left(X_i^{k+1}\right)$)
20：	for (j=0; j<維度; j++)
21：	$P_{i,j}^{k+1}=X_{i,j}^{k+1}$;
22：	end for
23：	end if
24：	if ($f\left(P_g^k\right)>f\left(X_i^{k+1}\right)$)
25：	for (j=0; j<維度; j++)
26：	$P_{g,j}^{k+1}=X_{i,j}^{k+1}$;
27：	end for
28：	end for
29：	counter++;
30：	while(counter>limit)
31：	得最佳解：$P_{g,j}^{k}$

圖 16 粒子群最佳化虛擬碼

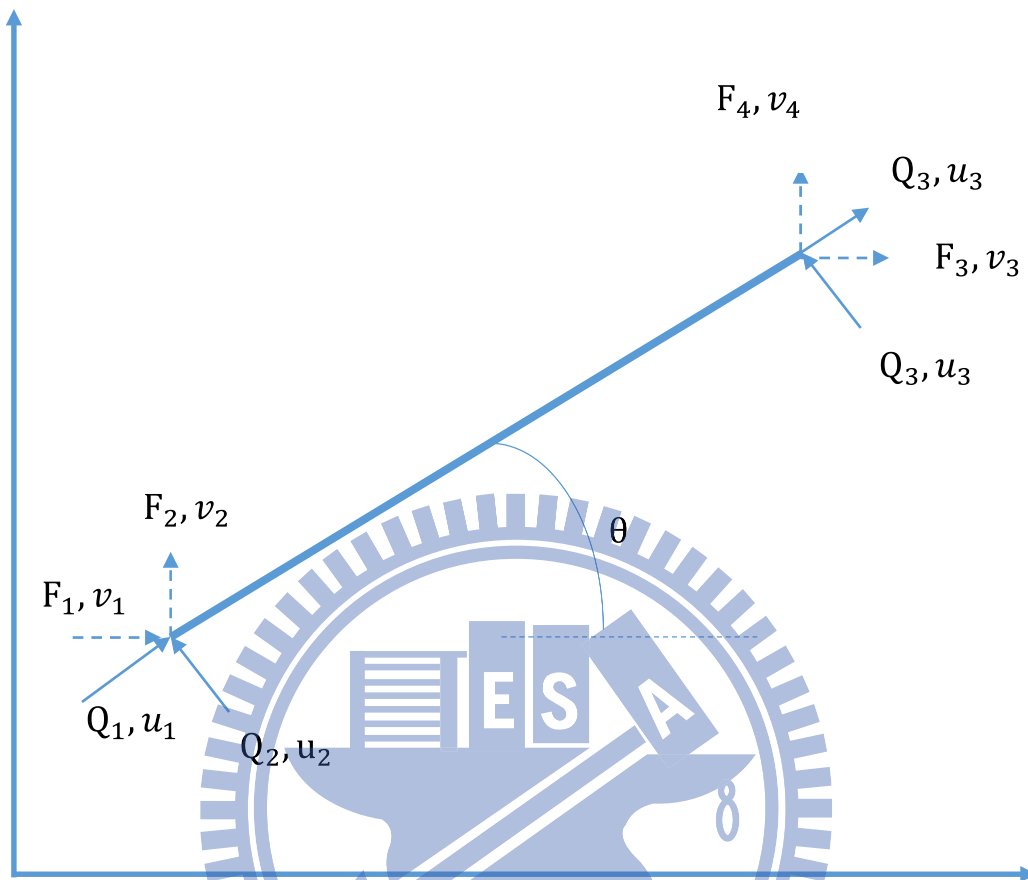


圖 17 二維桁架桿件局部座標和整體座標

說明：

$Q_1 - Q_4$：局部座標中的內力

$u_1 - u_4$：局被座標中的變形

$F_1 - F_4$：整體座標中的內力

$v_1 - v_4$：整體座標中的變形

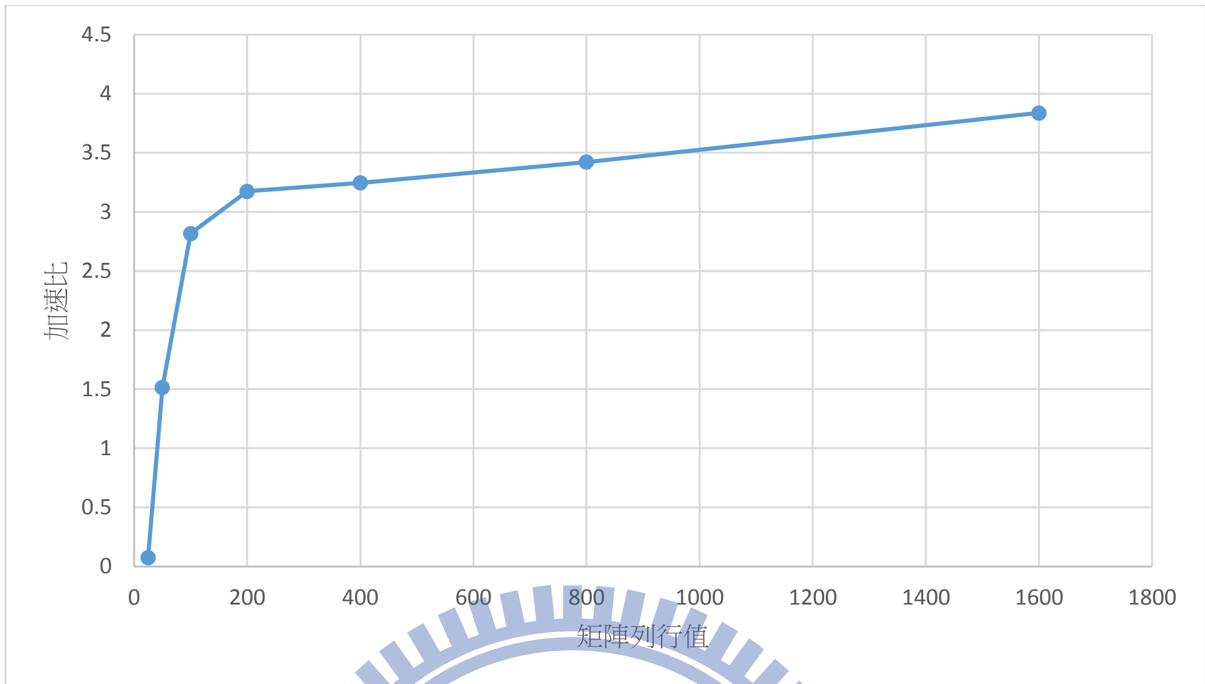


圖 18 矩陣乘法執行效能結果

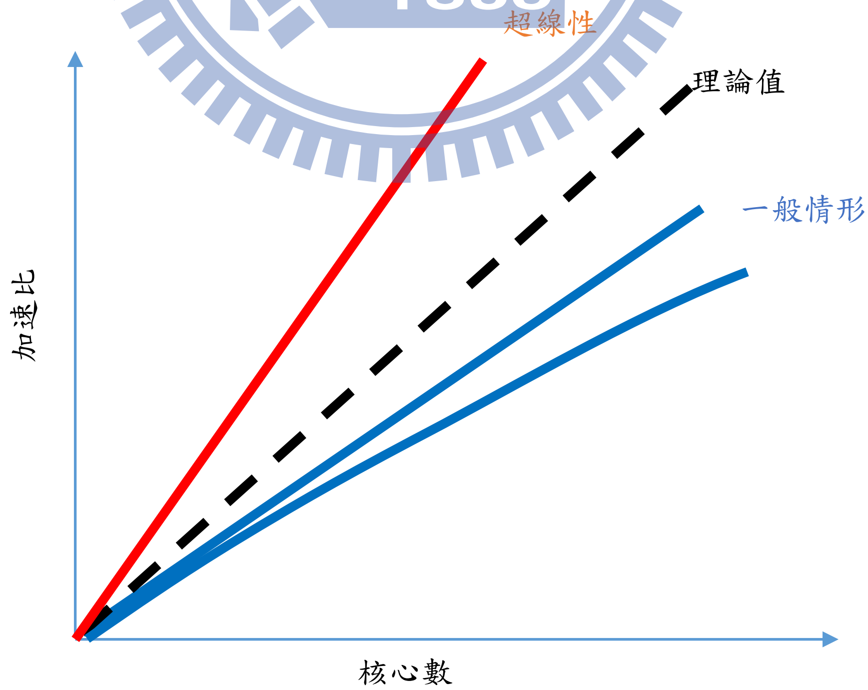


圖 19 核心數與加速比關係

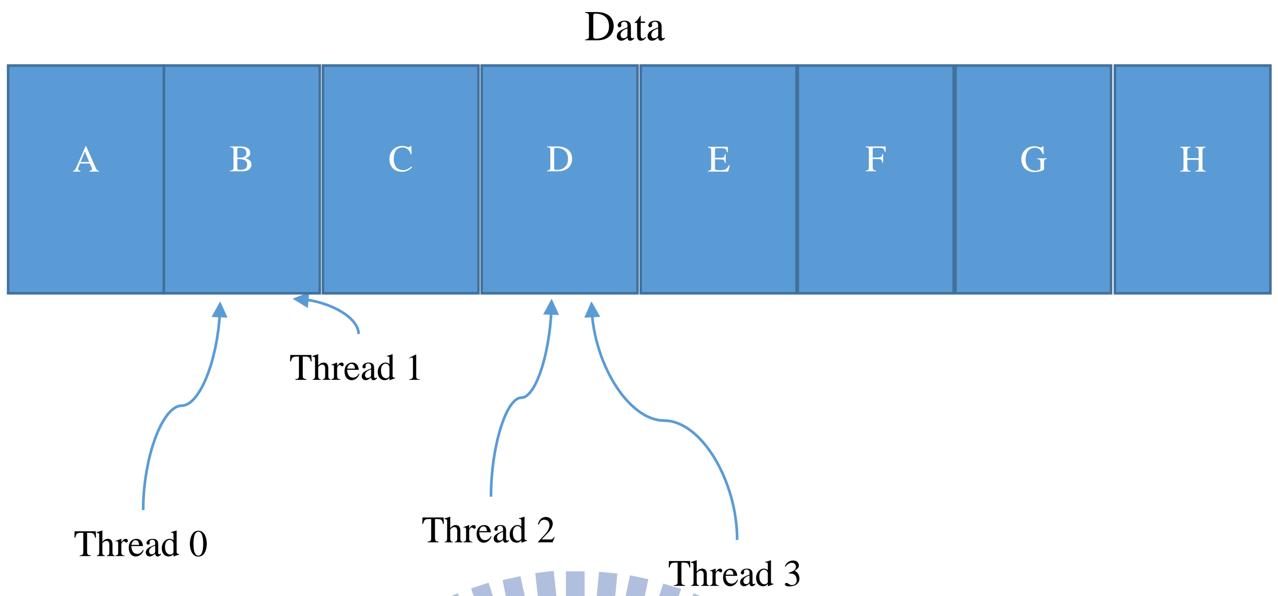


圖 20 多執行緒同時存取同一個記憶體位置

PSO

```
1： for (i=0; i<Total_of_particle; i++)
2：   for (j=0; j<dimension; j++)
3：     initialize：X^0_{i,j} 、V^0_{i,j} 、P^0_{i,j}
4：     calculate fitness function
5：   end for
6： end for
7： for (i=0; i<Total_of_particle; i++)
8：   if( f(P^0_g)>f(X^0_i) )
9：     P^0_{g,j}=X^0_{i,j};
10： end for
11： do{
12：   Rendon number:r_1, r_2
13：   # pragma omp sections
14：   {
15：     # pragma omp section
16：     {
17：       for (i=0; i<Total_of_particle/4; i++)
18：         (Kernel code)
19：       end for
20：     }
21：     # pragma omp section
22：     {
23：       for (i= Total_of_particle /4; i<2*
          Total_of_particle /4; i++)
24：         (Kernel code)
25：       end for
26：     }
27：     # pragma omp section
28：     {
29：       for (i=2* Total_of_particle /4; i<3*
          Total_of_particle /4; i++)
30：         (Kernel code)
31：       end for
```

1896

```
32：     }
33：     # pragma omp section
34：     {
35：       for (i=3* Total_of_particle /4; i<4*
       Total_of_particle /4; i++)
36：         (Kernel code)
37：       end for
38：     }
39：   if (f(P_g^k) > f(X_i^{k+1}))
40：     for (j=0; j<dimension; j++)
41：       P_{g,j}^{k+1} = X_{i,j}^{k+1};
42：     end for
43：   counter++;
44： while(counter>limit)
45： solution：P_{g,j}^k
```

圖 21 平行粒子群最佳化演算法

Kernel Code

```
1：     for (j=0; j< dimension; j++) {
2：       V_{i,j}^{k+1} = V_{i,j}^k + c_1 r_1 (P_{i,j}^k − X_{i,j}^k) + c_2 r_2 (P_{g,j}^k − X_{i,j}^k);
3：       X_{i,j}^{k+1} = X_{i,j}^k + V_{i,j}^{k+1};
4：     end for
5：     calculate f((X_{i,j}^{k+1})_{tid});
6：     if (f(P_i^k) > f(X_i^{k+1}))
7：       for (j=0; j< dimension; j++)
8：         P_{i,j}^{k+1} = X_{i,j}^{k+1};
9：       end for
10：    end if
11：    if (f(P_g^k) > f(X_i^{k+1}))
12：      for (j=0; j< dimension; j++)
13：        P_{g,j}^{k+1} = X_{i,j}^{k+1};
14：      end for
```

圖 22 粒子群最佳化核心

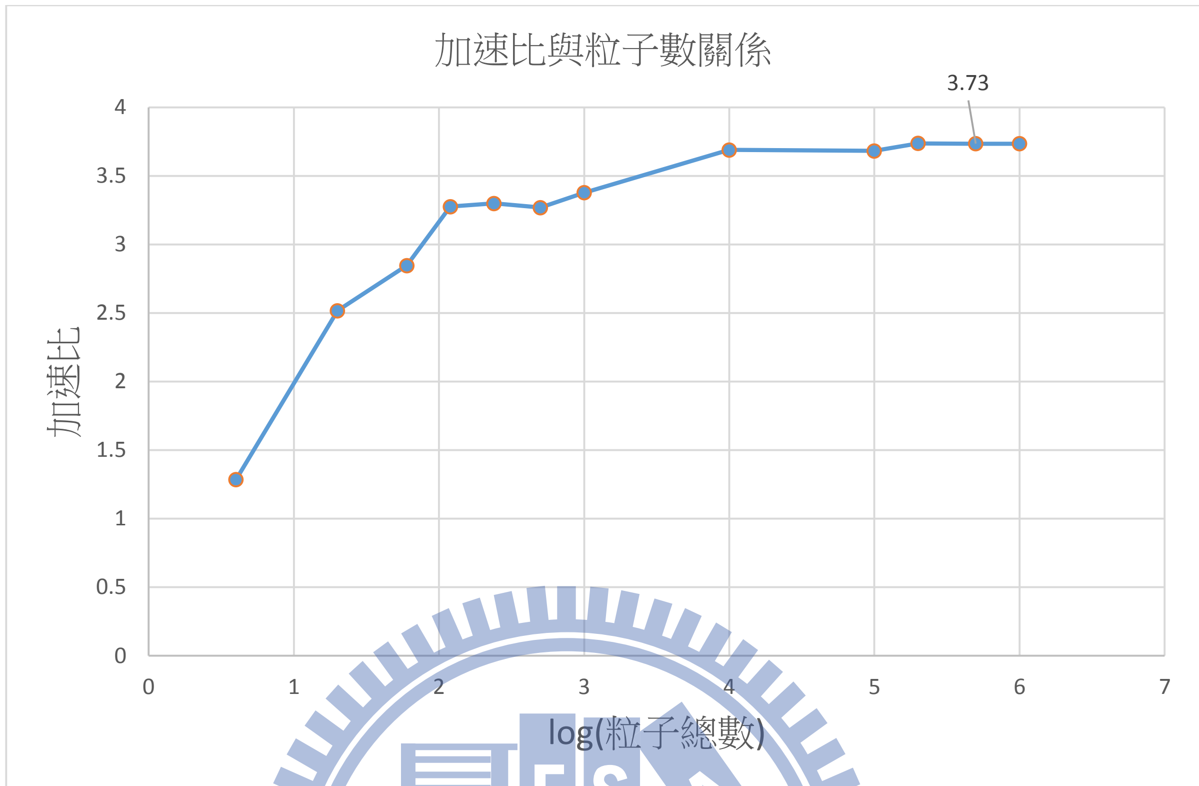


圖 23　30 個變數的最小平方和粒子總數與加速比關係

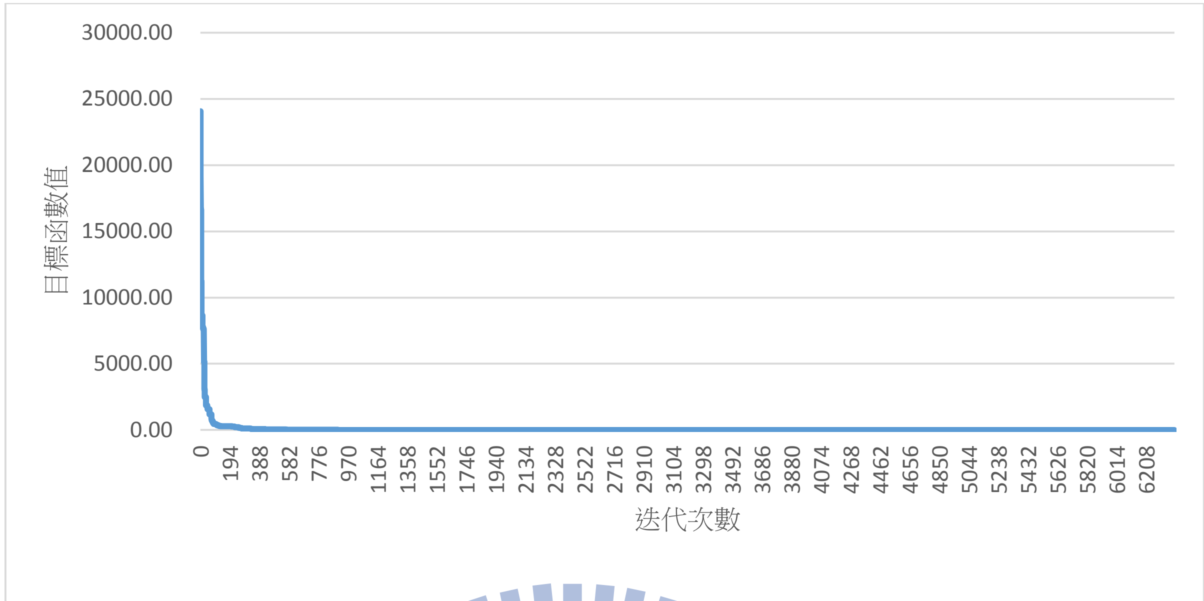


圖 24 4.2.1 節最佳化收斂情形

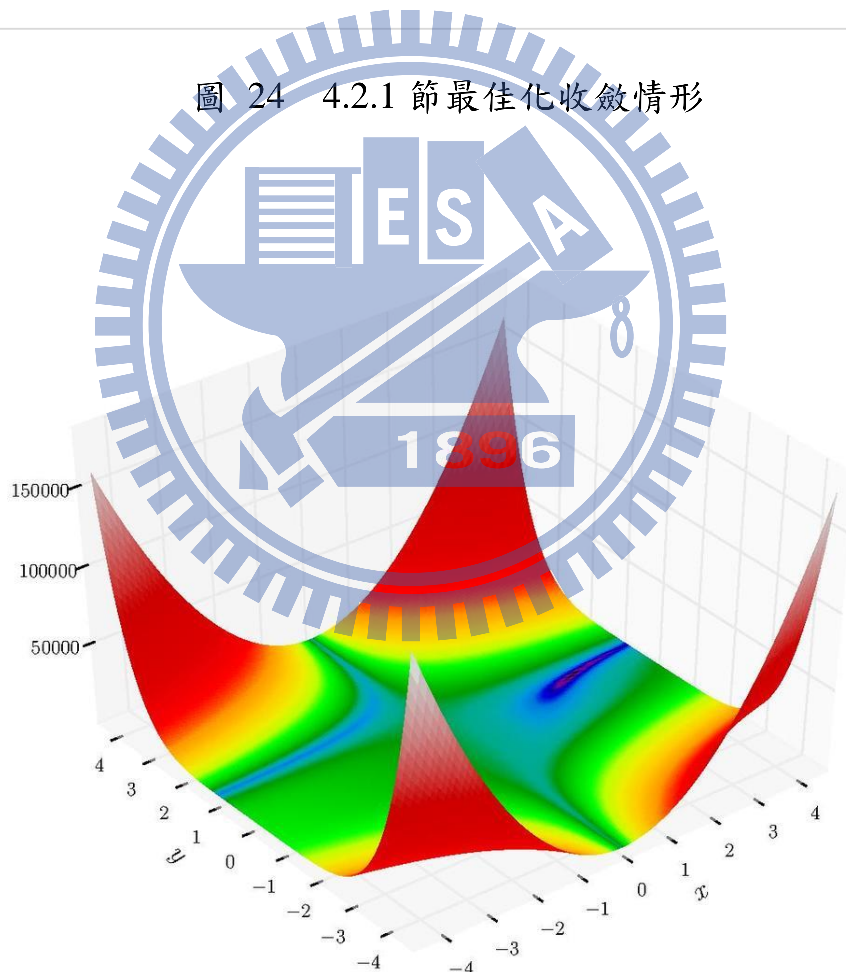


圖 25 Beale's function 繪製圖形

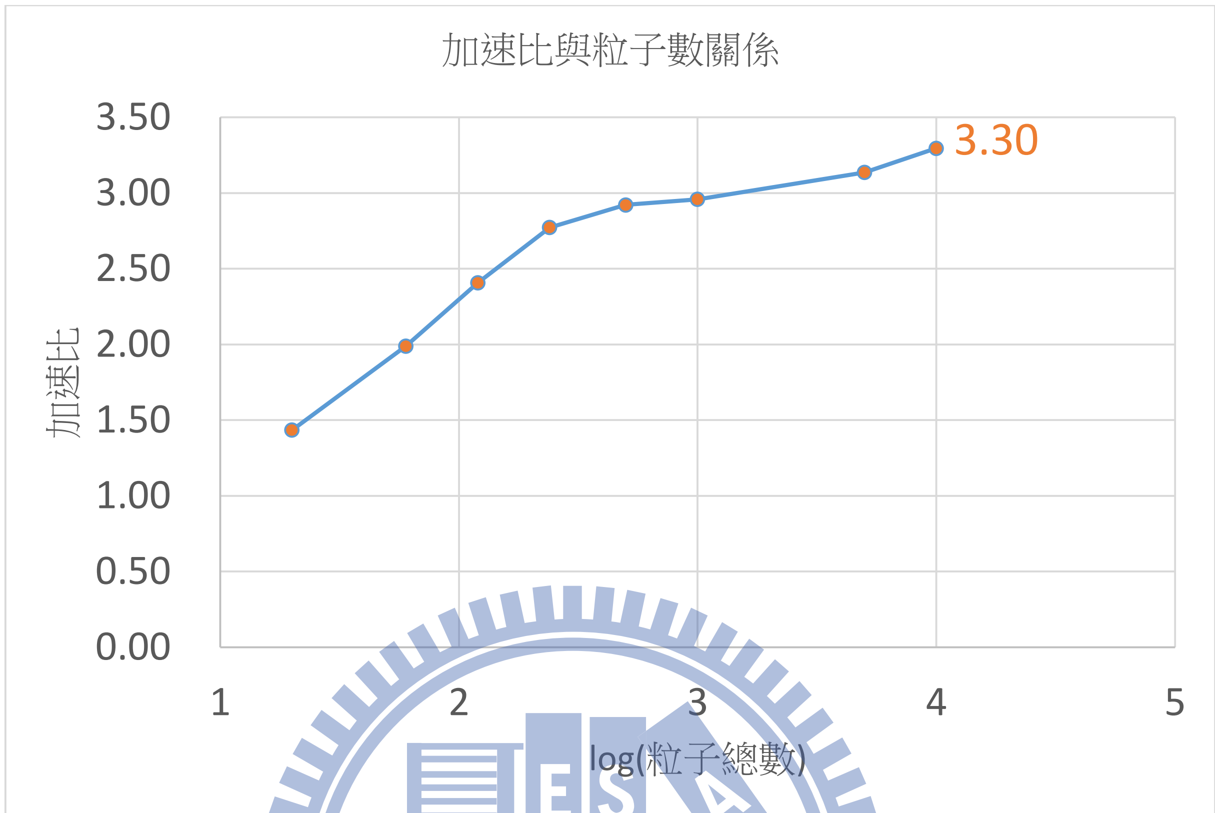


圖 26 Beale's function 粒子總數與加速比關係

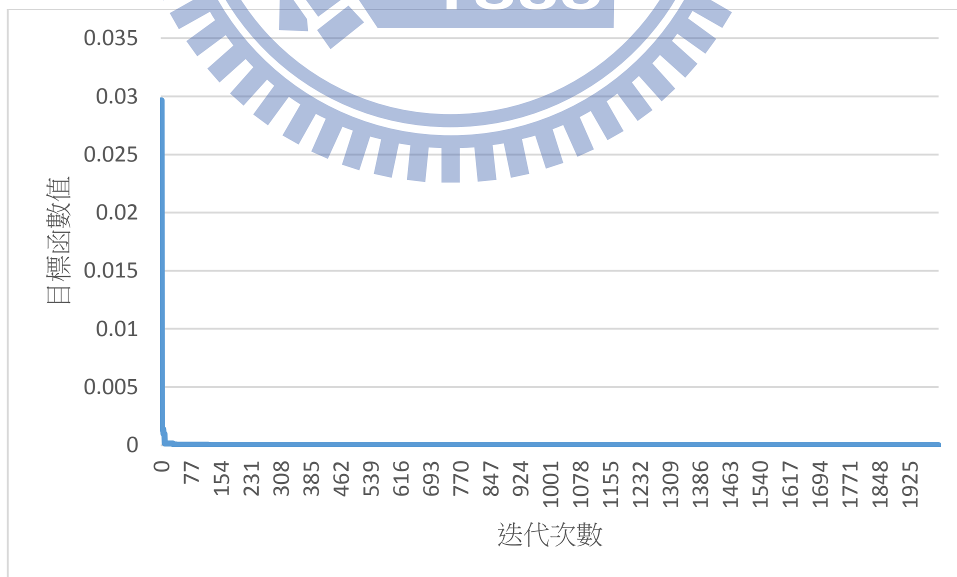


圖 27 Beale's function 收斂情形

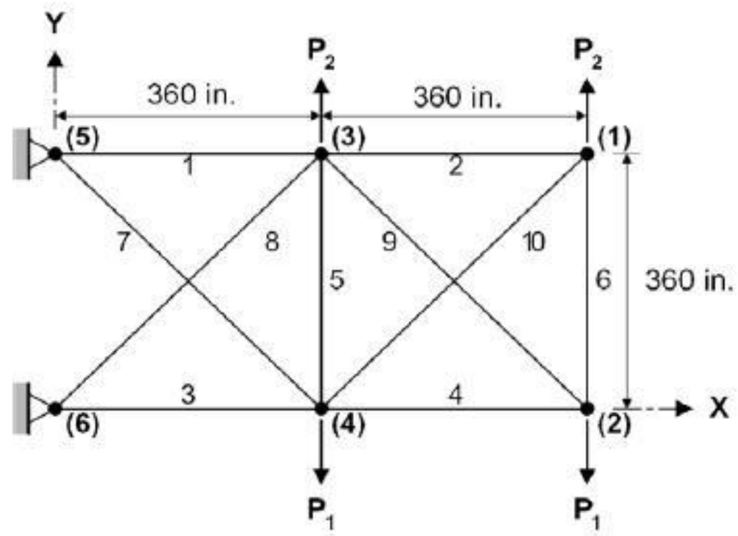


圖 28　10 支桿件桁架結構

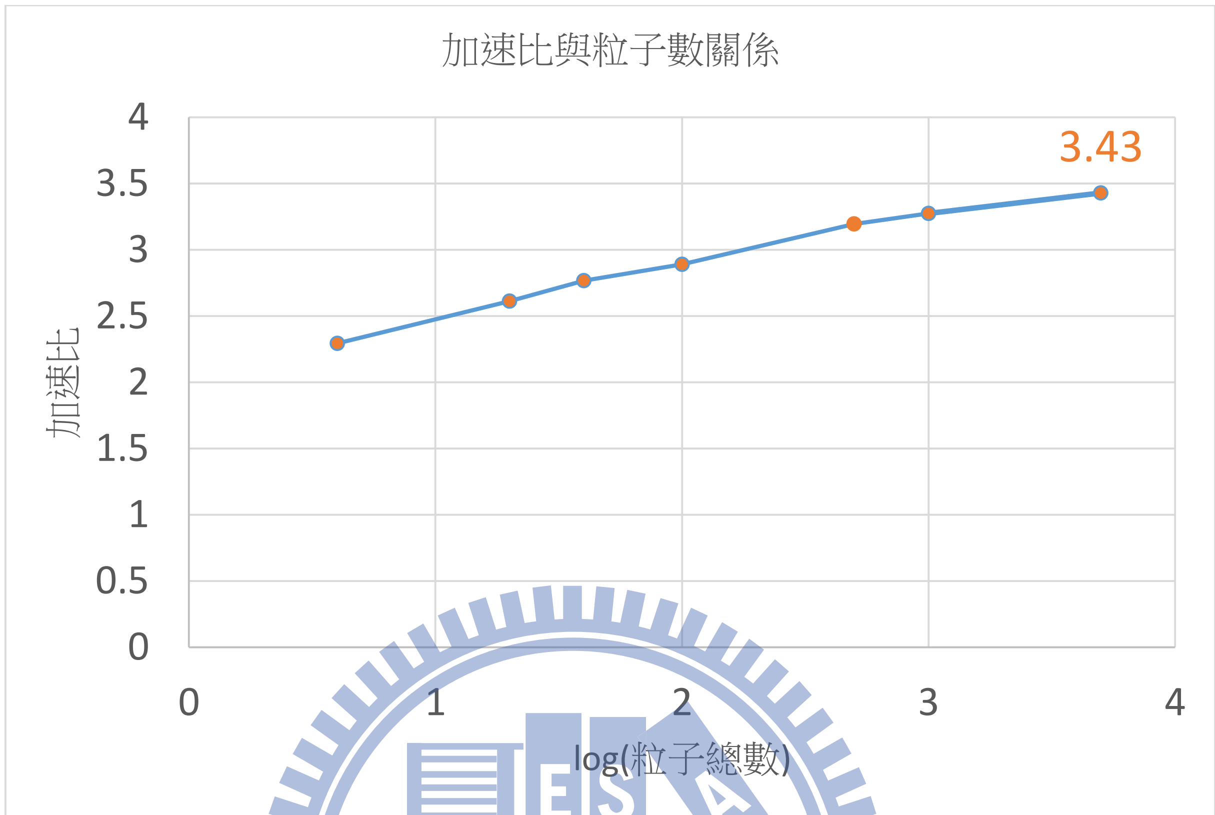


圖 29　10 支桁架加速比與粒子總數關係

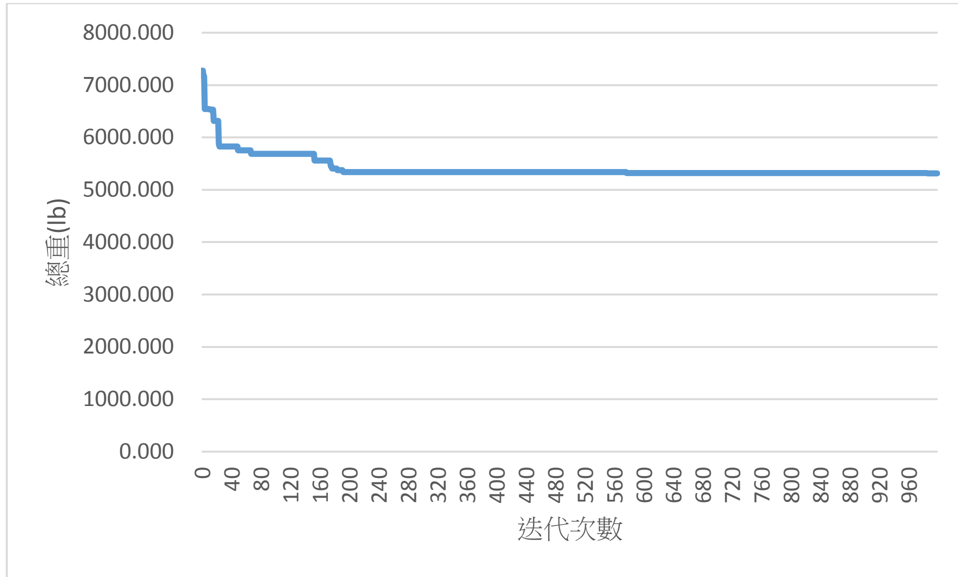


圖 30　10 支桿件桁架結構最佳化收斂過程

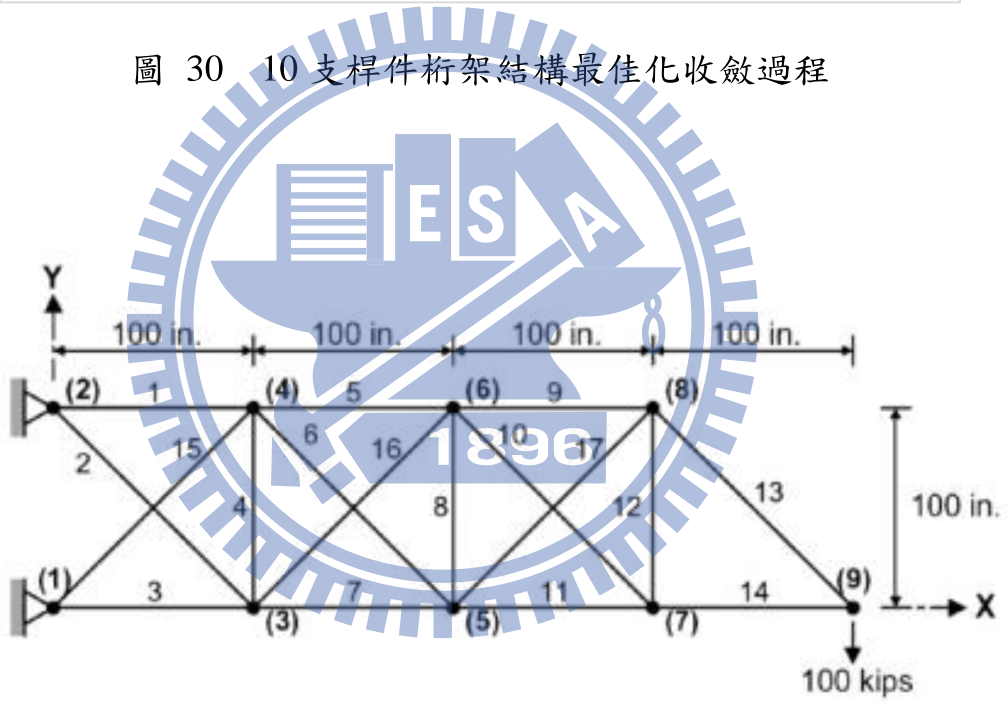


圖 31　17 支桁架結構

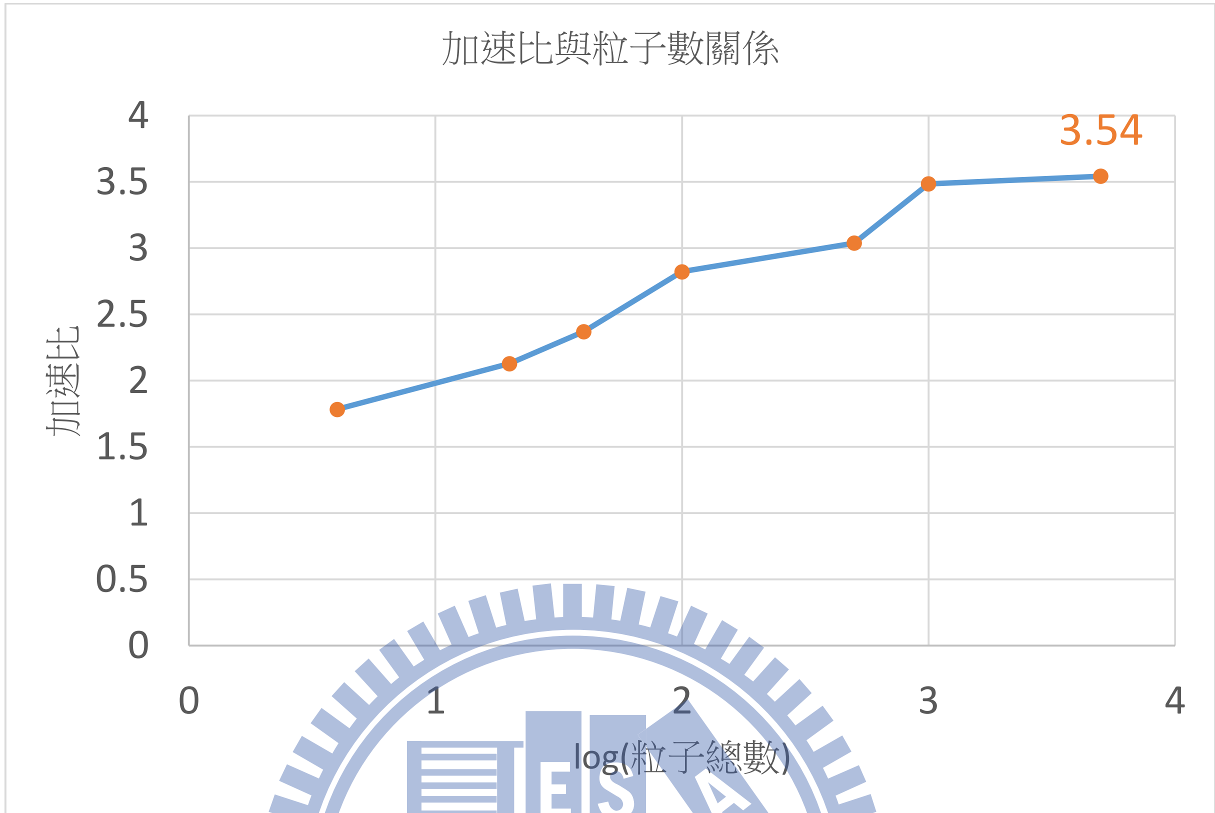


圖 32　17 支桁架加速比與粒子總數關係

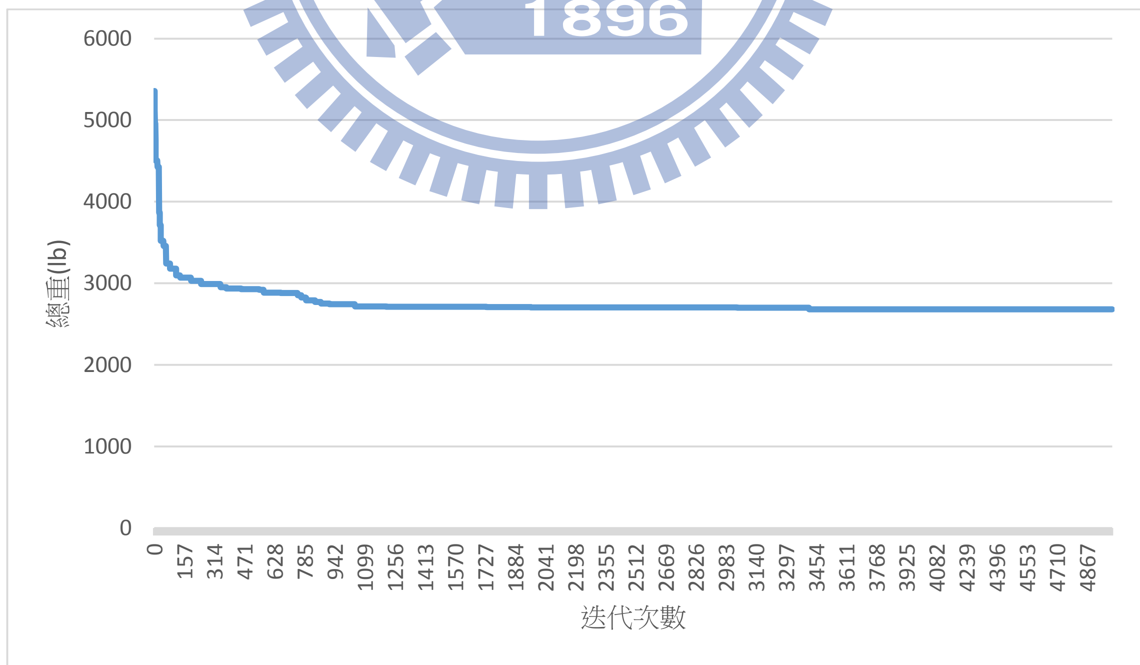


圖 33　17 支桿件桁架結構最佳化收斂過程

1896

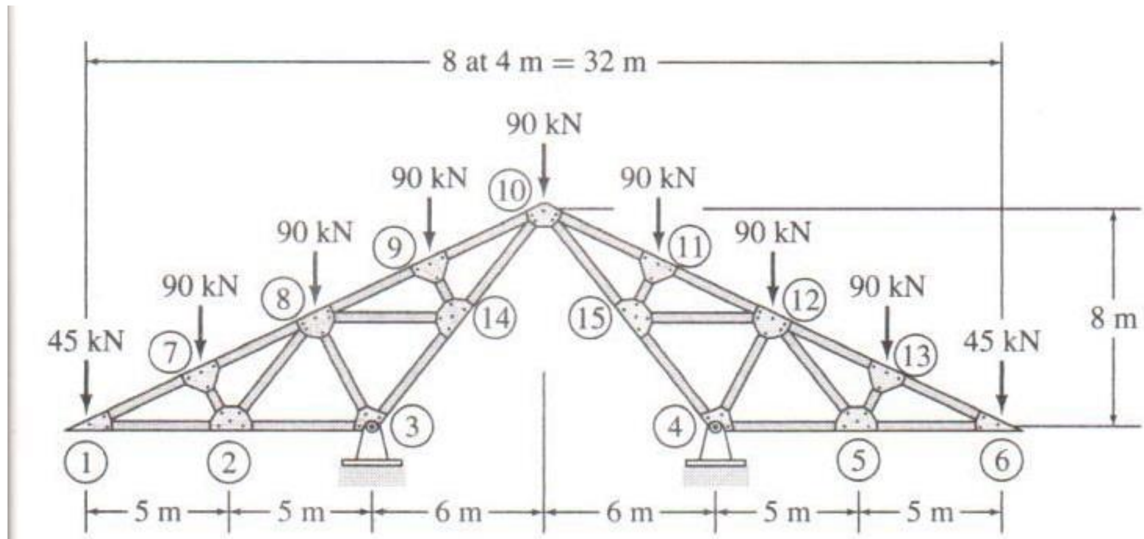


圖 34 26 支桁架結構

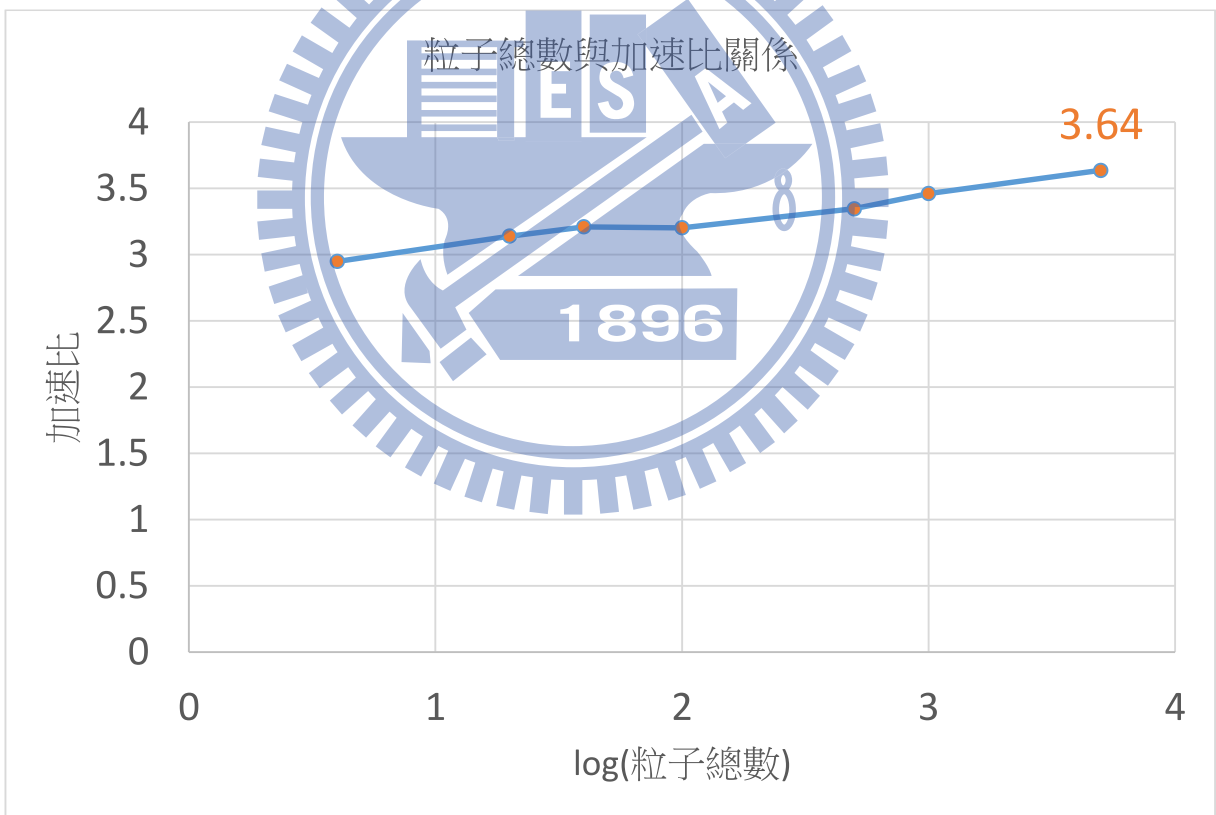
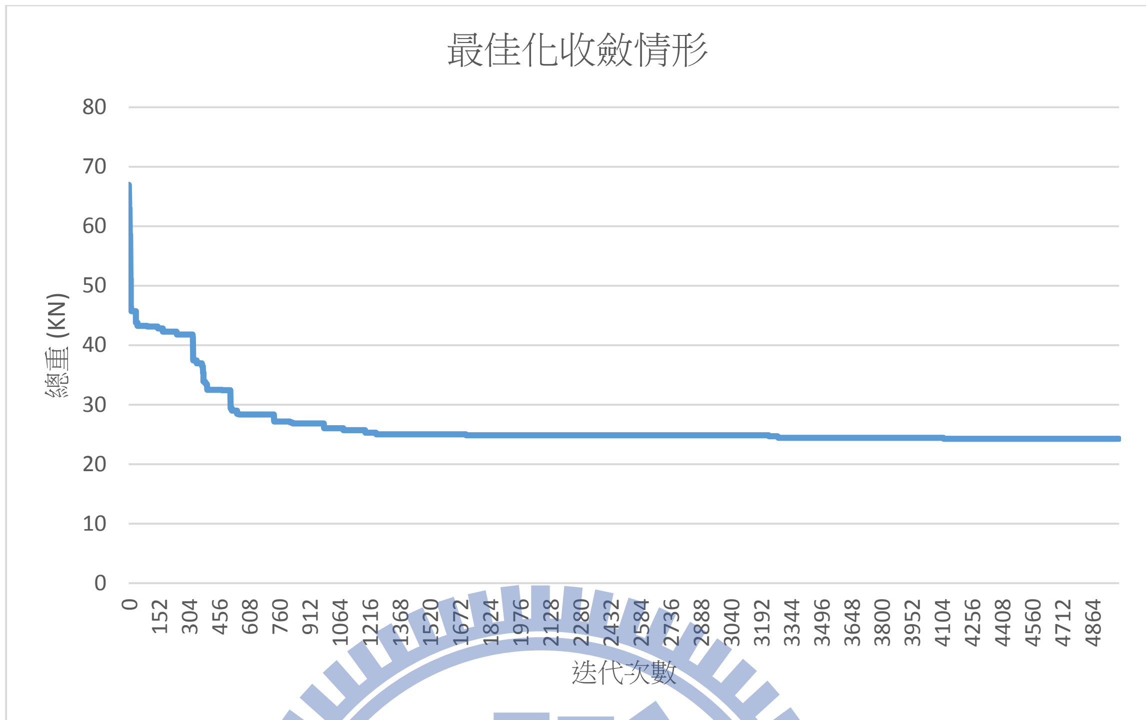


圖 35 26 支桁架結構粒子總數與加速比關係



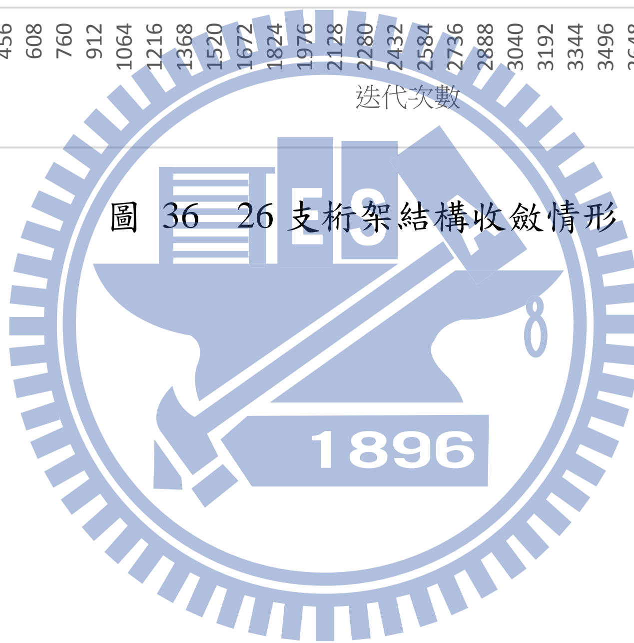


圖 36　26 支桁架結構收斂情形