東海大學資訊工程學系研究所

碩士論文

指導教授：林祝興 博士

Dr. Chu-Hsing Lin

基於島模型以及模擬退火法之 GPU 平行化基因演算法

Parallel Genetic Algorithms on the GPU Using Island Model and Simulated Annealing

研究生：李正傑

中 華 民 國 一 〇 五 年 六 月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 李 正 傑 所提之論文

基於島模型以及模擬退火法之GPU平行化基因演算法

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人 _____ 簽章

委 員 _____

_____

_____

_____

指 導 教 授 _____ 簽章

中華民國 105 年 6 月 21 日

中文摘要

目前對於 NP-hard 問題的解決方式，我們依然是利用找出近似解的演算法來降低其複雜度，雖然速度比起窮舉法要來的快，但是缺點是大部分的狀況下只能找出近似解。而基因演算法是一種隨機全域搜索和優化的方法，而最初的基因演算法有著許多樣的缺點，如過早收斂、容易掉進區域最佳解等問題。而後來出現了平行化基因演算法(PGA)來解決這樣的問題，目前在平行化基因演算法的領域上已經有非常多的研究了，也衍伸出了許多的演算模型。本篇的研究主要是利用 GPU 有著大量核心數的特性來找出並且優化適合 GPU 的 SIMD 架構的演算模型，並且配合平行化模擬退火法讓我們可以在平行化基因演算法上有著更好的效果。

關鍵詞：PGA、基因演算法、GPU、SIMD、模擬退火法、NP-hard 問題、島模型

ABSTRACT

To solve NP-hard problems, we can use algorithms for finding approximate solutions to reduce the complexity of the problems. Although this approach can come up with solutions much faster than brute-force methods, the downside of it is that only approximate solutions are found in most situations. Genetic algorithm is a global search heuristic and optimization method. Initially, genetic algorithms have many shortcomings, such as premature convergence and the tendency to converge towards local optimal solutions; hence many parallel genetic algorithms have been proposed to solve these problems. Currently, there exist many literatures on parallel genetic algorithms. Also, a variety of parallel genetic algorithms have been derived. This study mainly uses the advantages of the GPU, which has a large number of cores, and identifies better algorithms suitable for computation in single instruction, multiple data (SIMD) architecture of the GPU. Furthermore, the parallel simulated annealing algorithm is also adopted to enhance performance of the parallel genetic algorithm.

Keywords: Parallel computing, Genetic algorithm, TSP, GPU computing, Island model, Simulated annealing

致謝

兩年研究所的日子轉眼間就要結束了，這些日子以來有許許多的人們需要感謝，感謝有你們的陪伴，幫忙與諒解。

首先需要感謝的是在大學時就幫助我許多，從大學專題到研究所的指導老師 林祝興老師，願意讓我在研究所時嘗試這麼特殊的議題，真的非常的感謝老師一路上的包容與幫助。

李正傑 於東海大學研究所 資安實驗室　105 年 6 月

目錄

圖目錄

表目錄

方程式目錄

第一章　簡介

近年來在基因演算法的改良上已經有著非常多的進步了，而平行化基因演算法也是其中之一，但目前最多的仍然是利用多核心 CPU 進行平行化的演算，而近幾年的 GPU 也開始支持科學的運算以及處理，這使得我們可以將原本在 CPU 上運算的演算法搬到 GPU 上做更高平行化的運算，但由於 GPU 跟 CPU 的架構是不同的，所以在移植的過程中必須要改變演算法內相依性過高的部分，若是無法改變則這樣的演算法就無法在 GPU 上展現高效能的運算了。

目前的平行化基因演算法有許多的運算模型，較為知名的如 master-slave 模型以及島模型等，這兩種模型都是由好幾個族群群集各自獨立進行 GA 的運算，而差異就在如何交換族群個體以達到更優的表現。最初我們在平行化基因演算法時只是單純的將初始化、評價、選擇、交配以及突變移植到 GPU 的硬體架構上並且進行優化。雖然在同樣的族群數的數量上疊代的速度上快了上百倍，但依然無法解決過早收斂以及容易掉進區域最佳解的問題。所以對於這兩個問題我們參考了了島模型將大數量的族群數切成好幾塊區塊各自進行數次的 GA 運算最後在利用全域選擇操作進行族群個體的交換。這樣的算法的好處除了可以改善容易掉進區域最佳解的問題之外還可以利用 shared 記憶體的存取來加速 GPU 上的運算，並且

在將模擬退火法與基因演算法結合以改善基因演算法在過早收斂上的問題。我們將利用 TSP 問題來當作基準。對於 TSP 問題目前已經有許多研究發展出了多種幾何算法來解決 TSP 問題，由於本研究是著重在基因演算法本身效益上的開發，所以對於 TSP 問題我們只有利用基因演算法進行運算以比較出 CPU 以及 GPU 之間的差異以及各項參數對於平行化基因演算法的影響。

第二章　背景知識與相關文獻

在過去幾年的平行化基因演算法在硬體上的表現仍然都只受限於多數量的 CPU，而 CPU 在核心的數量方面至今仍遠低於 GPU 的核心數。其中的原因在於 CPU 是 MIMD 架構而 GPU 是 SIMD 架構，簡單說即是 CPU 的核心架構較 GPU 複雜，所以在一個單位的空間內能放入的核心數量就會大量的少於 GPU 了。GPU 發展到了現在只要我們的演算法能夠符合以下 2 點，就能夠在 GPU 的運算架構上加速。

1. 資料間的獨立。(解釋：當兩個變數資料在同一時間內同時做運算並且不會改變任何運算結果，則稱為資料間的獨立)
2. 所有的資料皆需要大量但不複雜的運算，並且是同樣的演算方式。

如果一個演算法符合以上兩點，那麼我們就可以說，這個演算法是適合在 GPU 上進行運算的。

2.1 GPU 平行運算

過去 GPU 被製造出來的目的主要是為了處理圖形的渲染以及顯示，而近年來 GPU 漸漸的被製造成除了圖形顯示之外還可以進行科學運算的裝置，最早被稱為 GPGPU(General-purpose computing on graphics processing units)。由於 GPGPU 的出現讓 GPU 擁有了科學運算的能力，也就是我們可以在 GPU 上進行編程。而後出現了 OpenCL 以及 CUDA 這兩個用於進行 GPU 編程的 API，其中的差異在於 CUDA 只能用於 NVIDIA 公司旗下的產品。除了 API 的差異以外，GPU 的硬體也跟過去大不相同。GPU 平行運算與 CPU 平行運算的差異在於 GPU 除了擁有 CPU 無法比擬的大量核心數之外，GPU 的架構是屬於 SIMD 的架構。這代表了 GPU 在科學計算上一次只能運行一樣工作，雖然擁有大量的核心數量，但同一時間我只能處理一件事情，無法像 CPU 那樣可以進行多工的分配。所以可以知道適合 GPU 的運算通常都是單純且大量的演算，並且 CPU 在 GPU 進行運算的同時仍然可以擁有處理運算的能力。這讓我們的演

算法除了單純的拆分成 SIMD 架構的平行化之外，更可以利用 CPU 以及 GPU 所組成的 HAS 架構進行更進一步的優化。

2.2 TSP 問題

為了能夠讓基因演算法有個基準來量化數據，我們選了 TSP 問題來進行效能比較上的量化依據。

TSP(Travelling Salesman Problem)是數學領域中非常著名的一個問題。這個問題假設有一個旅行者要拜訪 n 個城市，他必須選擇他要走的路徑，路徑的限制是一個城市只能拜訪一次，最後要回到原來出發的城市，求出所有可能路徑裡面的最小路徑值。

TSP 問題是一個排列組合優化的問題， 舉一個例子，假設有 4 個節點，以$node_0$為起點，首先我們定義問題域為式(2.1)

$$X=\{node_0\ ,node_1\ ,node_2\ ,node_3\} \quad (2.1)$$

並且我的目標函數為式(2.2)

$$Minimize\ f(x) = dist(node_0, x[0]) + \sum_{i=0}^{2}[dist(x[i], x[x+1])] + dist(x[3], node_0) \quad (2.2)$$

那麼其實我們可以看到要解這樣的公式所要耗費的運算資源是非常龐大的，並且我們也無法簡單的解出 Minimum f(x)，有鑑於此我們認為使用基因演算法來解決這樣的問題是非常適合的。

第三章　GPU 架構下的平行化基因演算法設計

目前在基因演算法的平行運算上已經有許多種的運算模型，如 master-slave 模型、島模型等，這些的模型利用了平行化可以進行同時運算的特性將大量的族群分區並且各區都各自擁有不同的隨機搜尋域。而這些運算模型主要就是改良過去基因演算法會衍生出過早收斂、容易收斂至區域最佳解問題的方法。而這些運算模型的平行運算部分用於 CPU 其實就是利用多核 CPU 的多工特性，不管切分出來的區塊有多少，就是將正在閒置的核心塞進還未被進行運算的區塊即可。而在 GPU 的平行化則與 CPU 大不相同，由於 GPU 的運算核心沒有多工的能力，並且由於 GPU 與主記憶體的溝通必須透過 Peripheral Component Interconnect Express(電腦匯流排 PCI 的一種，它沿用了現有的 PCI 編程概念及通訊標準)來進行傳輸。若是使用的演算法過於讓 GPU 與主記憶體過度溝通的話，所有的運算時間將會花在記憶體的存取上。所以，我們打算先將基因演算

法的所有步驟都利用 GPU 進行運算，最後再結合平行化基因演算的優化模型。

3.1 基於 SIMD 架構的平行化演算法

首先基因演算法分為五個步驟:

1. 初始化族群：

   隨機產生 n 個染色體 (n 由使用者決定)。

2. 染色體評價：

   利用適應函數計算所有染色體的適應值。

3. 染色體選擇&複製：

   依每個染色體的適應「選擇、複製」染色體。

4. 染色體交配：

   對留下的染色體進行交配的動作。

5. 突變：

   對留下的染色體進行突變的動作。

而這五個步驟除了評價階段以外都必須使用到亂數，而平行化的亂數產生方式其實就是將亂數的種子進行位移，因為我們知道電腦產生亂數的方式是利用一個極大的數列並且塞入種子(seed)來進行此數列的查表。所以利用 NVIDIA 所提供的 *curand.kernel* 將每個執行緒 ID 當作該執行緒的種子，這樣一來每個執行緒就可以各自產生

獨立的亂數域了。而評價的部分則是每個執行緒各自運算自己的染色體的評價值後存入全域記憶體。

3.2 SIMD 架構下的平行化選擇與交配演算法

在平行選擇(Selection)的部分，基因演算法之中 Selection 有著許多的方式，其中最著名的便是輪盤法，但輪盤法本身對於 SIMD 架構的平行化運算有著一個很大的缺點。首先我們可以看到輪盤法在分配機率區域時所用的式(3.1)。

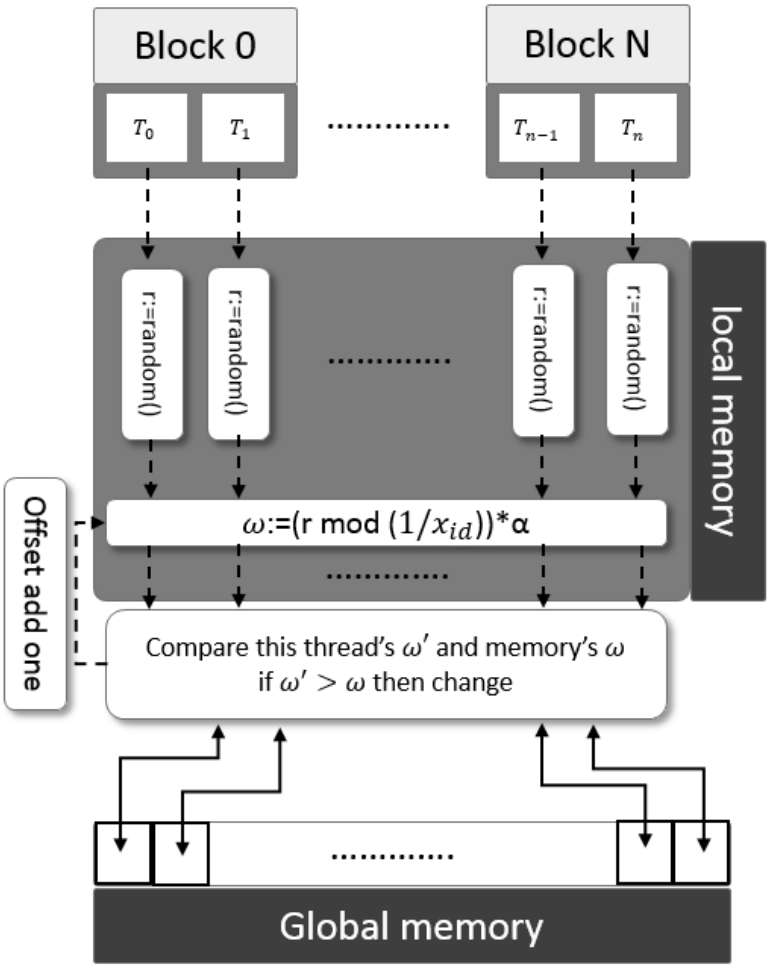
定義：{x} = 族群集合, i=染色體代號;

$$\text{Probability}(x_i) = \frac{x_i}{\sum_{k=1}^{n} x_k} \times 100\% \qquad (3.1)$$

雖然$\sum_{k=1}^{n} x_k$本身只需要被計算一次即可，但由於先前介紹過，我們的目的是將基因演算法全部只由 GPU 進行運算來避免過於頻繁的與主記憶體以及 CPU 進行資料交換，所以雖然這個算式只需要被算一次，但對於 SIMD 架構的 GPU 來說是每個執行緒 s 都算過一次，這樣除了浪費了效能之外還會拖垮運算速度。所以我們利用了式(3.2)進行效能上的改良，其中α為權重精度，因為需要進行除餘(mod)運算，所以必須要是整數運算。 定義：$\omega =$ 染色體權重, $\alpha =$ 權重精度;

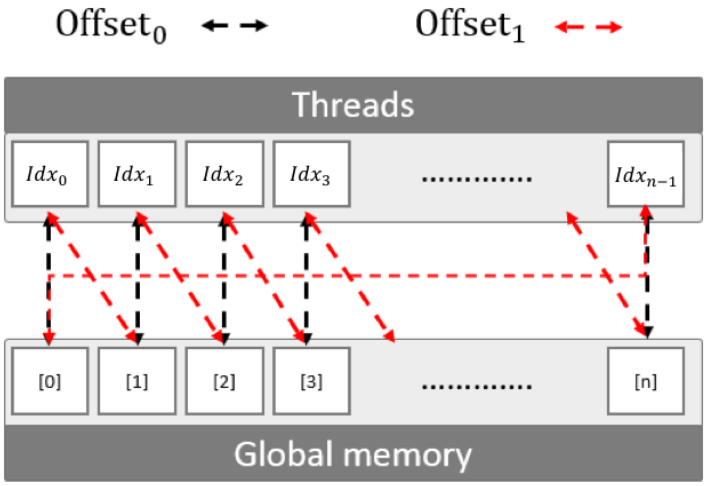
$$\omega_i = Random(thread_{id}) \bmod \alpha * \frac{1}{x_i} \qquad (3.2)$$

利用式(3.2)，我們可以運算出這條染色體的權重。而在複製過程中每一塊記憶體會挑選權重最高的進行複製，複製到所設定的族群數便停止複製，這部分在 GPU 架構下的平行方式如圖一。



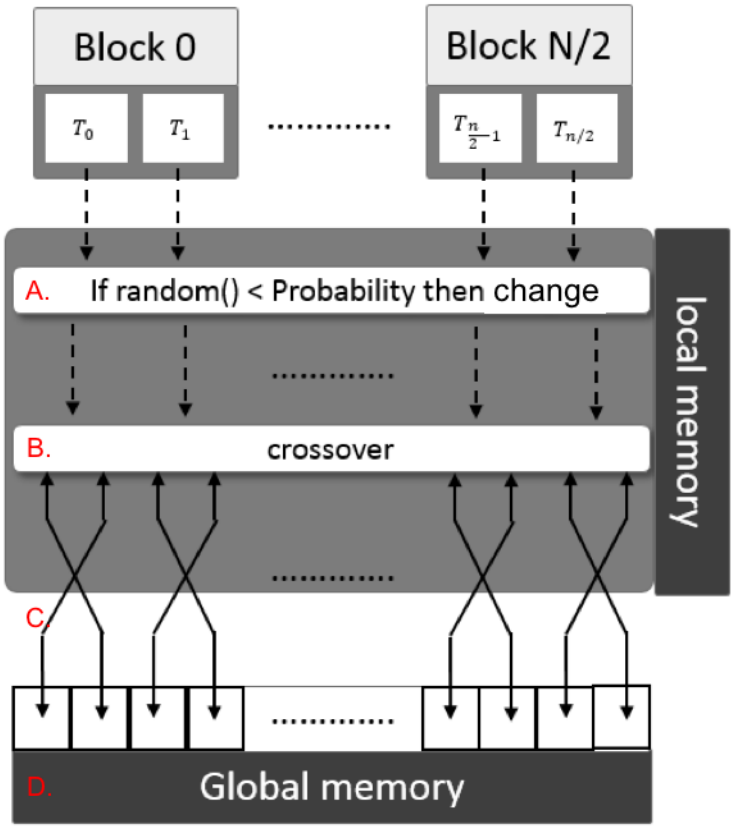
圖一 GPU 架構下之平行化基因演算法流程圖

首先利用式(2)算出各執行緒所處理的染色體權重後，便對相對於自己 ID 的記憶體進行比較操作; 若是記憶體所儲存的比較小的話，便將此執行緒的 ID 塞進相應這塊記憶體的 ID 的複製池內，結束後所有的執行緒的相應記憶體位置之 offset 都加一。當 offset 加上執行緒 ID 超過總族群數的值時便與族群數數量的值進行除餘運算找出下一個要存取的記憶體位置，然後再進行一次式(3.2)的運算，一直重覆到 offset 數值等於族群數的大小便停止此核心的運算(圖二)。這樣的算法可以確保複製池的每一格都有進行完全的輪盤選擇而不會因為平行化的關係產生執行緒 s 在存取記憶體的時候所產生的衝突造成資料存取不一致，並且根據式(3.2)的算法，就算是評價較差的染色體也有機會可以進入複製池，以確保基因的多樣性來防止過早收斂。



圖二 平行化選擇操作記憶體存取方式

在來就是染色體交配的平行化分析。我們採用的交配方式是相鄰交配，主要的原因在於 GPU 在記憶體上存取的限制，我們可以先看到圖三的 GPU 架構平行化染色體交配操作的演算流程。



圖三 GPU 平行化交配操作

A. 這是一行由電腦操作機率的運算方法，機率(Probability)的值越大則代表 random()操作越有機會滿足此條件式並且交配操作進行交換(change)就越有可能發生，反之。

B. 根據上一步的機率運算現在每個執行緒內將在交配(crossover)操作時進行染色體的交換與否。

C. 此步驟表示若是執行緒要進行染色體的交換時將會以一個執行緒操作兩個染色體進行交換，並且兩個染色體的位置是相鄰的。

D. 交換後的染色體將會以一個執行緒相對兩個記憶體位置的方式存進全域記憶體內。

我們將執行緒 s 的數量從與族群數相同切到族群數的半數，主要的原因是一個執行緒要處理兩條染色體的交配過程，而這樣的一個架構仍然可以讓 GPU 的記憶體存取以 half-warp 的速度進行。

(half-warp：一次的存取量將為半個 warp(16 個記憶體位置)進行存取)

3.3 SIMD 架構下的平行化突變與模擬退火法

突變部分平行化其實就只是單純的利用每個執行緒各自的亂數域對自己所操作的染色體進行單點或是多點突變。當然這樣的突變法是最陽春的方式，而我們利用了改良後的突變法－模擬退火法。模擬退火法顧名思義就是參考了冶金學在鋼鐵退火時的演算法，其原理在於當一個材料被加熱至高溫後其內部的原子便能離開原來的位置並隨機移動，其目的在於當溫度慢慢冷卻時隨著原子可以變動的範圍變小讓其慢慢收斂至比原先還要更小的位置讓材料的晶粒

排列更為緊致。而模擬退火法就是模仿這一點來優化隨機搜尋的演算法讓其能夠更快的收斂至較佳的解答，而模擬退火法的算法我們可以看到表一。

表一 模擬退火法演算法

```
定義：s =原子狀態, e=原子能量, emax=能量停止點, k=評估次數, kmax=最大評估次數  sn=鄰近的原子狀態, de=鄰近原子的能量，Temp = 溫度函數;
inpute: s0, emax, kmax
output: s
s := s0
k:=0
e := E(s)
// 若還有時間（評估次數 k 還不到 kmax）且結果還不夠好（能量 e 不夠低）則：
while k < kmax && e > emax
// 隨機選取一鄰近狀態 sn
sn := neighbor (s)
// de 的能量為 E (sn)
de := E(sn)
// 決定是否移至鄰近狀態 sn
if random() < P(e, de, Temp(k/Kmax)) then
// 移至鄰近狀態 sn
    s:= sn
    e:= de
// 評估完成，次數 k 加一
k :=k+1
return s
```

首先我們會有一個原子的變數，而 e 代表著這個原子目前的能量，接下來我們隨機改變這個原子的變數至鄰近的狀態，此時利用式(3.3)計算改變此原子的能量的機率，再利用亂數產生 0 至 1 的數字，若是亂數所產生的數字小於式(3.3)所產生的數字便改變此原子的變數。

定義：$e =$ 適應值, $e' =$ 突變後的適應值, Temp = 溫度;

$$P\left(e, e', Temp\left(\frac{k}{Kmax}\right)\right) = -\frac{e' - e}{Temp} \times 100\% \qquad (3.3)$$

而 Temp 則會隨著時間下降，最後降至 0 時便無法再進行原子變數的變更。而結合於基因演算法則是取代突變的部分，變更後的演算法如表二。

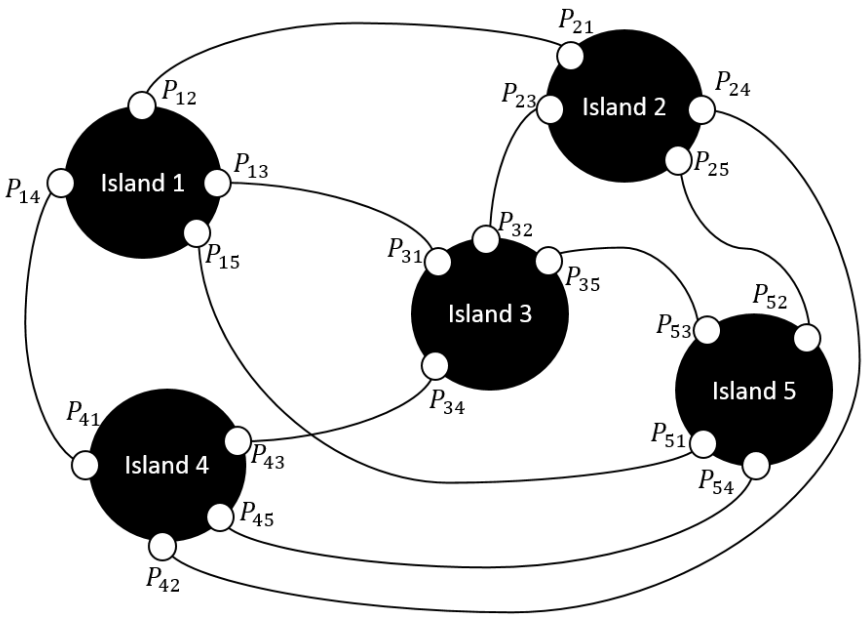
表二 模擬退火法結合基因演算法

```
定義：E =適應值, d_E=突變後的適應值, Temperature = 目前的溫度值，x=染色體，evaluation(): 適應值函數，Mutation()：突變函數, exp(): 以自然數為底數的指數函數
inpute: x
output: Mutation(x) or x
E:=evaluation(x);
d_E:=evaluation(Mutation(x));
deata:=d_E – E;
//若是改變的能量不小於等於零並且利用自然數底數的指數函數算出來的機率來決定是否要改變此染色體
if deata>0 and exp(-deata/ Temp) > random() then
    E:=d_E;
```

我們將原子的變數取代成染色體，並且能量則代表著此染色體的評價值，雖然結合了模擬退火法加快了基因演算法收斂至最優解速度，但所需要調整的變數也多了起始溫度以及下降率，但當參數調整得宜的話，其優化程度將會比原先的突變還要好。

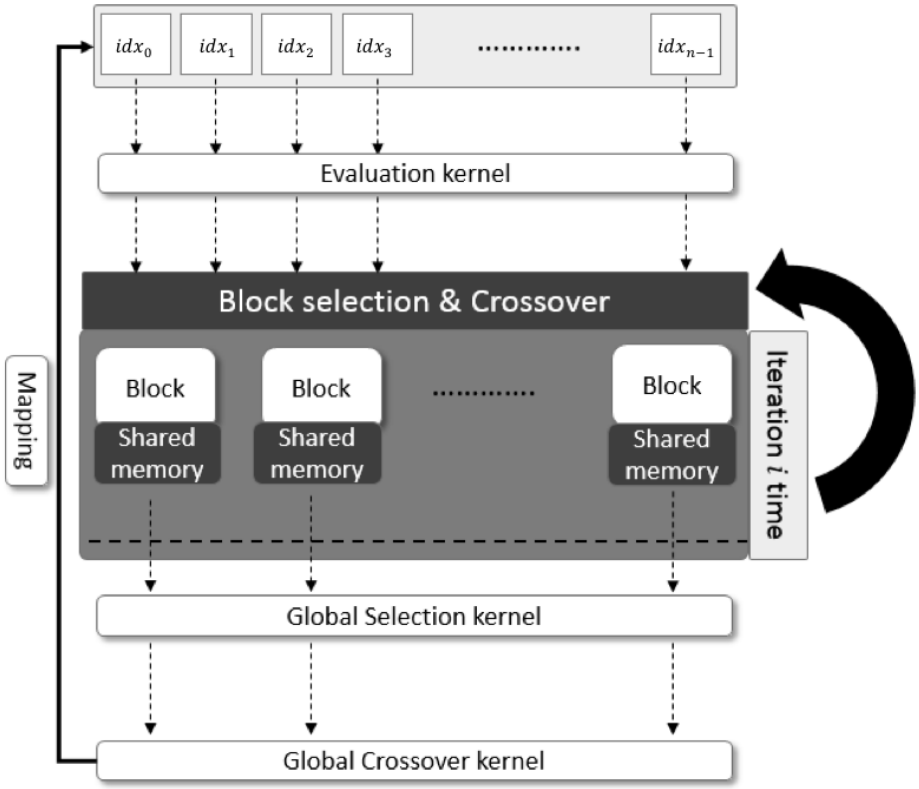
第四章　結合島模型的平行化基因演算法

在原先的部份我們已經將基因演算法的所有操作都在 SIMD 架構上的平行運算上實現了，但即使如此我們也只有增加其在疊代運算上的速度，並沒有解決基因演算法的一些固有的缺點，如過早收斂至區域最佳解等問題。而島模型(圖四)也因為有比 serial single 族群數模型還要好的表現[1]，所以我們打算以島模型為基礎進行這一部分的改良。



圖四 島模型

島模型主要就是將族群切分成好幾塊的區塊，其作用在於每個獨立的區塊(Island)個別運作著基因演算法產生各自的隨機搜尋域，而在染色體的遷移方面每個區塊之間擁有著一定的機率，適應值較高的染色體將會有較高的機率進行遷移。在本研究我們是利用區塊經過 i 次的疊代各自進行基因演算法之後利用一次的全域選擇操作來讓所有區塊的基因進行一次交換，當然適應值較佳的染色體也容易遷移至各個區塊。而這樣做的目的在於讓收斂到較不佳解的區塊可以利用評價較好的染色體讓整個隨機搜尋域找出更佳的解答。而這樣的演算法在平行化上只需要在選擇以及交配操作時進行區塊的分離，其餘的仍然可以沿用先前的平行化模式，其在 SIMD 架構下的平行運算過程可以看到圖五。



圖五 以島模型為基底的 GPU 平行化分區基因演算法

我們將總數量的族群數切分成 M 個區塊，所以每個區塊的族群數數量將會有$\frac{n}{M}$個[1]。而每塊的區塊在疊代至我們所設定的 i 次值(圖五)之前皆會各自進行獨立隨機搜尋的基因演算法，當疊代數量達到 i 值後便會進行一次的全域選擇操作以及交配操作來讓各區塊之間交換染色體，之後便再重複進行各區塊的獨立基因演算法。

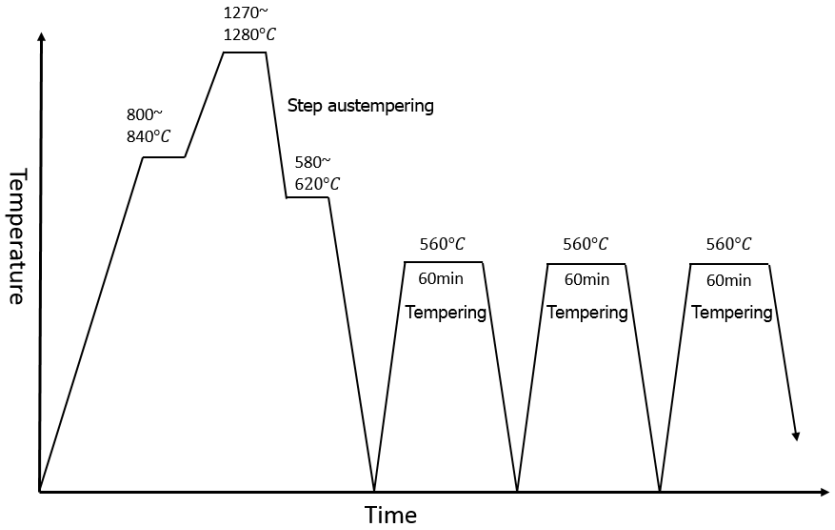
第五章　模擬退火法之下降率研究

雖然模擬退火法是一個能夠優化隨機搜尋演算法的一種良好的算法，但其中模擬退火法的困難點在於下降率的參數調整，下降的太快會容易卡在區域最佳解，而下降的太慢則會遲遲無法收斂，所以我們測試了兩種的下降率方式，首先根據式(5.1)我們可以得出最簡單的指數下降法。

定義：Temp=溫度, ROD = 下降率;

$$\text{Temp} \coloneqq \text{Temp} \times \text{ROD} \qquad (5.1)$$

但其中我們在放進基因演算法測試時會發現，當我染色體的長度越長，其設定的單一下降率在最後收斂時有非常大的機率會卡在區域最佳解。所以接下來第三種方法我們參考了冶金學的球化退火法並且以此為模型設計出了不同於先前的下降率運算方式，而當中我們參考了週期球化退火法(圖六)。



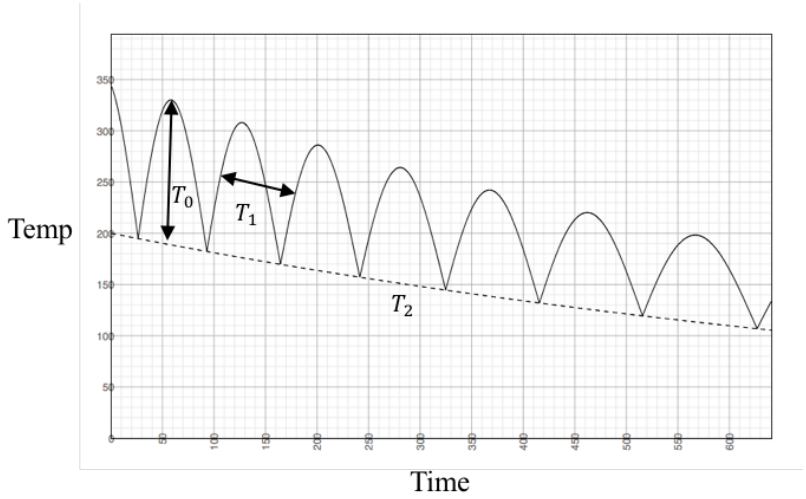
圖六 球化退火法

以此模型為參考我們設計出了式(5.2)。

定義：T_0=振幅, T_1=波長, T_2=斜率, $\mathrm{ROD} = \{ROD_0, ROD_1, ROD_2\}$,

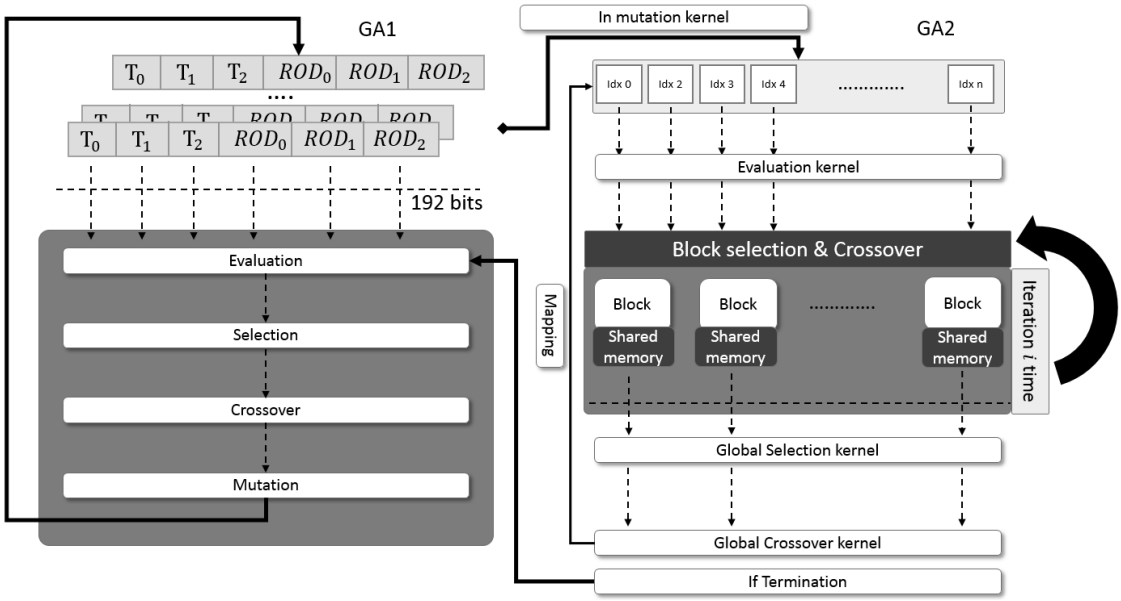
Temp = 溫度;

$$\mathrm{Temp} = T_0 \times ROD_0 \times \left|\cos T_1 \times ROD_1\right| + T_2 \times ROD_2 \quad (5.2)$$



圖七 球化退火法的數學模型

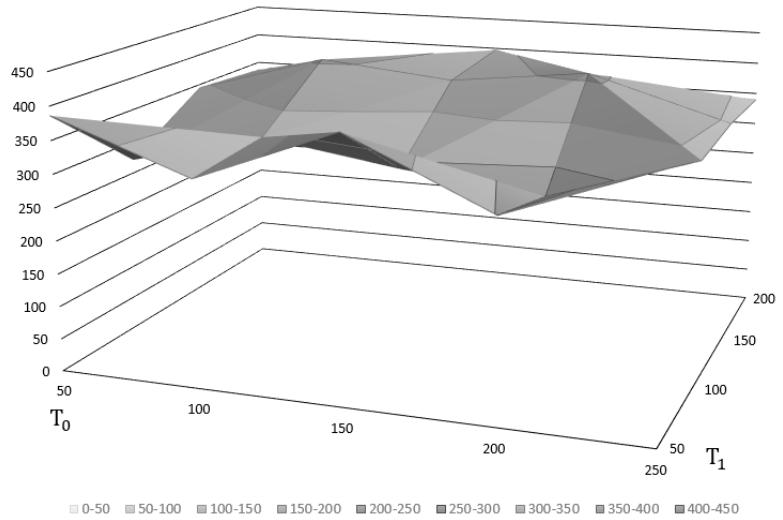
這樣的下降率算法主要是解決當基因演算法掉進一個區域最佳解可是溫度卻又下降到無法提供跳出此區域的狀況時所使用的一種方法。而關於參數的設定方面也是以人工的方式進行，但因為變數只有T_0, T_1, T_2以及ROD_0, ROD_1, ROD_2，所以在某種程度上要比多項式的下降率算法還要好設定。而在未來我們可以利用雙層的基因演算法來找出此下降率算法的參數，其中流程圖如圖八。



圖八 利用雙層基因演算法結構來優化退火法的參數之演算流程

由於每條染色體就代表著一次的下降率參數，並且適應值是利用GA2 進行多次的疊代運算而得到的，所以在族群數的部分不可能太

大。而當我們找到適合解決這個染色體長度的參數時，那們未來當基因演算法又再次遇到同樣染色體長度的問題便可以直接套用此下降率的參數以加快找出最佳解的速度。當然這樣的想法我們會在未來進行實作，而本篇的研究並未使用到此種技術，所以關於這方面的參數部分仍然都是利用人工的方式進行調整。接下來我們便開始進行這兩種下降率在同一個染色體長度的基因演算法下的收斂率比較。首先對於式(5.1)的下降率部分，相對於式(5.2)的部分是在式(5.2)的$T_2 \times ROD_2$ 這個式子，所以在進行比較實驗時這兩個參數將會設定的跟式(5.1)一樣以確保球化退火法相較於原本的單項式下降率算法是否有明顯的優化。實驗環境為 TSP 擁有 256 個節點，並且在五分鐘之內所跑出來之相對誤差率比較。首先將參數調整成 Temp=300;ROD=0.999 進行實驗，此下降率在球化退火法各個下降率的相對誤差率為圖九，在此下降率的單項式下降率算法之相對誤差率為 407.8804%。



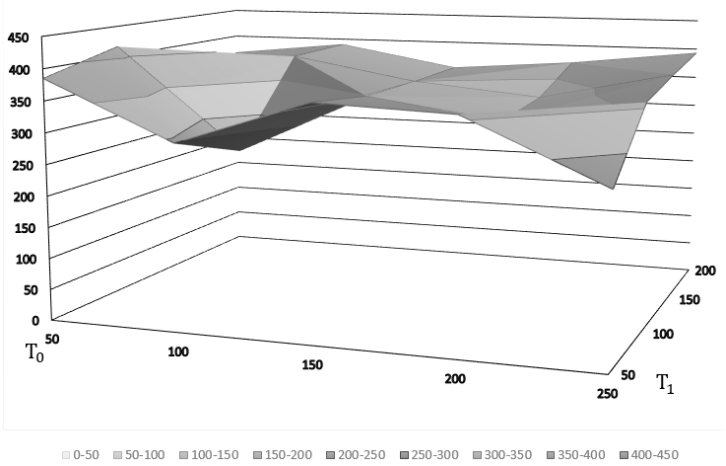
圖九 基礎下降率為 0.999 之球化退火法的各頻率與振幅的相對誤差率(ROD =ROD_2 =0.999; Temp=T_2=300)

從圖九我們可以觀察到當$T_1 = 100$時(也就是波長=100)，其收斂率會明顯小於其他數值(無論T_0的值為何)，所以影響收斂率的關鍵就在於T_1的值。

表三 基礎下降率為 0.999 之球化退火法的各頻率與振幅的相對誤差率

		T_1			
T_0		50	100	150	200
	50	386.2167	271.5436	344.2957	336.2516
	100	311.4254	323.2916	405.2371	357.8908
	150	394.6275	292.1579	384.4916	396.9173
	200	298.4466	268.2143	409.0585	351.9743
	250	379.3976	338.7134	345.4107	339.2736

Temp=300;ROD=0.9999 的相對誤差率為圖十，在此下降率的單項式下降率算法之相對誤差率為 346.7588%



圖十 基礎下降率為 0.9999 之球化退火法的各頻率與振幅的相對誤差率(ROD =ROD_2 =0.9999; Temp=T_2=300)

雖然圖十並沒有圖九來著明顯，但觀察起表四就可以明顯發現當$T_1 = 100$時，其相對誤差率依然低於T_1的其他值，所以由圖九以及圖十就可以證明這套公式最主要的關鍵除了斜率T_2再來就是T_1了。

表四 基礎下降率為 0.9999 之球化退火法的各頻率與振幅的相對誤差率

		T_1			
T_0		50	100	150	200
	50	385.0921	415.9965	321.2917	367.6405
	100	294.9293	250.6498	386.5825	389.6105
	150	365.4384	349.1226	349.9237	352.4873
	200	357.4735	319.7697	330.7815	368.6747
	250	263.3465	357.7516	376.6454	392.7839

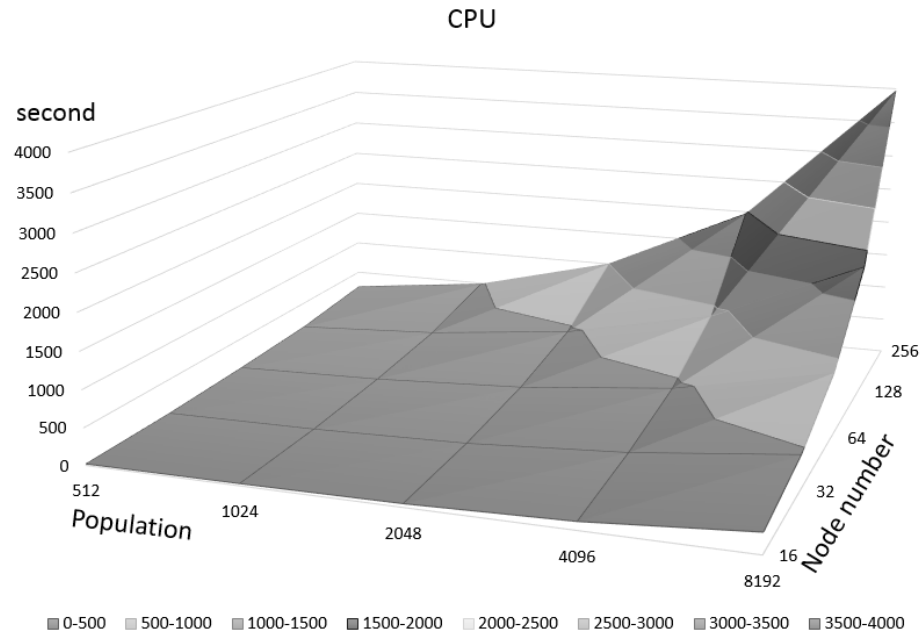
從這兩項的實驗結果我們可以看到，在單項式下降法加上球化退火法，可以有效的補足當溫度下降到一定的程度時仍保有一定的機率可以跳出區域最佳解。

第六章　結果與討論

在本次的實驗我們 CPU 的部分使用了:Intel(R)Core(TM)i7 CPU 4770K@3.5GHz; 記憶體:16.00 GB;而 GPU 使用了:Nvidia Titan Black; 作業系統使用:Linux Ubuntu 14.04(64bit)。CPU 單線程的程式使用了此網站(http://simulations.narod.ru/)上的基因演算法搜尋 TSP(使用 Matlab)。本次的實驗將本篇所提出的 SIMD 架構平行化基因演算法結合島模型並且再結合改良的模擬退火法與單線程 CPU 運算的基因演算法進行比較。 並且本次的實驗只單純使用基因演算法來運算 TSP，並沒有特別針對 TSP 進行優化，因為本次研究的目的在於發展出良好的基因演算法基礎，而 TSP 只是本次研究的基準而已。

6.1 族群大小與疊代速度

對於 CPU 基因演算法以及 GPU 基因演算法，首先可以比較的是在每個數量的族群數疊代 2000 次，並且配合不同數量的節點進行比較。首先我們量化了 CPU 在各個族群數下對應不同數量的節點所花費的時間。如圖十一與表五可以發現隨著族群總數以及節點數的增加，CPU 所花費的時間有明顯成長的趨勢。

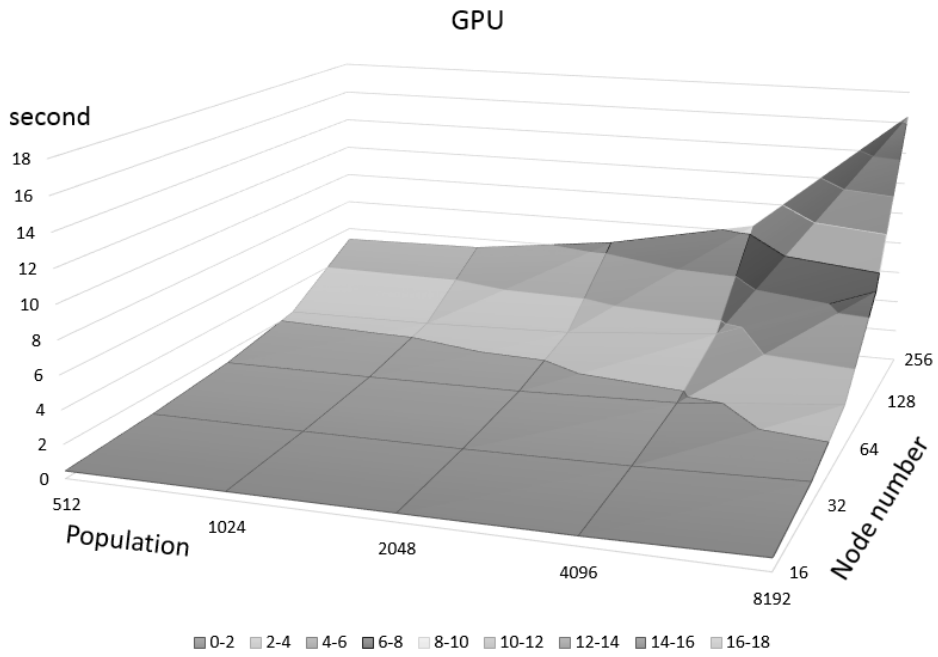


圖十一 CPU 族群數數量與 node 數量所需運算的時間量(疊代 2000 次，時間單位為秒)

表五 CPU 族群數數量與 node 數量所需運算的時間量(疊代 2000 次，時間單位為秒)

		Nodes				
		16	**32**	**64**	**128**	**256**
族群數	**512**	20	25.95	63.25	120.15	251.32
	1024	28.01	46.43	115.39	232.43	481.84
	2048	47.03	101.46	224.33	454.25	1000
	4096	109.62	233.87	438.87	892.00	1969.5
	8192	270.75	465.64	885.5	1807.75	3964

目前可以看到 CPU 在基因演算法上的運算消耗其實是非常高的，往往要跑出一個結果就必須耗費大量的時間進行等待。以下我們可以看到 GPU 的結果。



圖十二 GPU 族群數數量與 node 數量所需運算的時間量(疊代 2000 次，時間單位為秒)

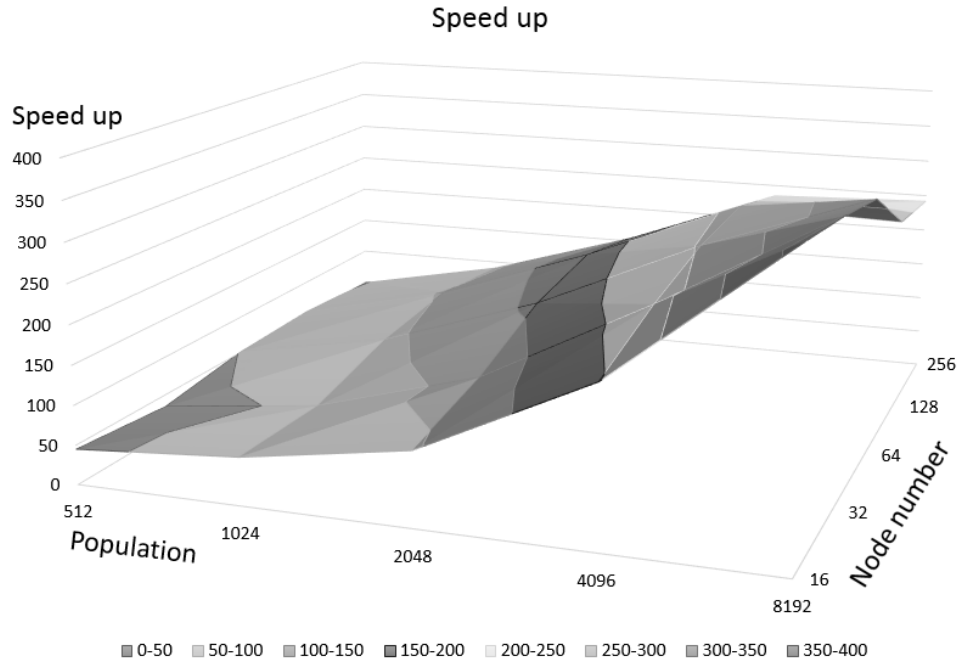
由圖十二可以觀察出以同樣的節點(Node)數來說，就算族群數(Population)以 2 的倍數成長，但由於 GPU 平行化的運算，其所花費的時間不會以 2 倍的時間成長。所以比起圖十一的 CPU，非常明顯的 CPU 只要族群數成長 2 倍，其花費時間也會成長 2 倍。不過 GPU 的核心數也是有限的，所以在本實驗可以看出當族群數從 4096 成長到 8192 時 GPU 所花費的運算時間相較於其他的族群數成長了

很多，所以當族群數超過 4096 後 GPU 所花費的運算時間也會呈現 2 的倍數成長。並且我們也可以觀察到當族群數為 4096 時便是本研究所使用的 GPU 所能平行化的極限了。

表六 GPU 族群數數量與 node 數量所需運算的時間量(疊代 2000 次，時間單位為秒)

		Nodes				
		16	**32**	**64**	**128**	**256**
Population	**512**	0.445	0.758	1.245	2.171	5.202
	1024	0.465	0.789	1.299	2.265	5.355
	2048	0.502	0.857	1.412	2.574	6.488
	4096	0.557	1.017	1.731	3.401	8.395
	8192	0.734	1.326	2.694	7.058	16.399

從 GPU 的實驗數據我們可以發現到，GPU 利用本篇所提出來的演算方式可以大幅的降低運算所要花費的時間，而 GPU 以及 CPU 的加速比如下圖十三。



圖十三 GPU 與 CPU 的時間加速比-比較圖(加速比公式：CPU 花費時間/GPU 花費時間)

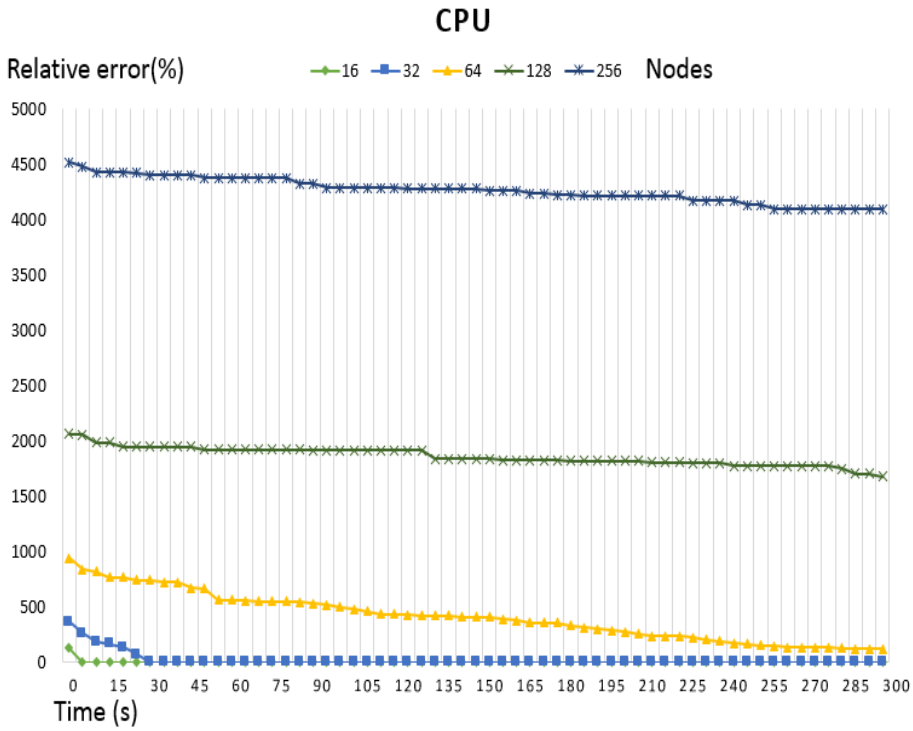
表七 GPU 與 CPU 的時間加速倍數比-比較表

		Nodes				
		16	**32**	**64**	**128**	**256**
Population	**512**	44.94	34.23	50.8	55.34	48.31
	1024	60.23	58.84	88.82	102.61	89.97
	2048	93.68	118.38	158.87	176.47	154.13
	4096	196.8	229.96	253.53	262.27	234.6
	8192	368.86	351.16	328.69	256.12	241.72

可以看到 GPU 在族群數(Population)越大時其加速比會越來越高，當加速比不斷的在成長時就代表 CPU 已達極限但 GPU 仍未到達極限。

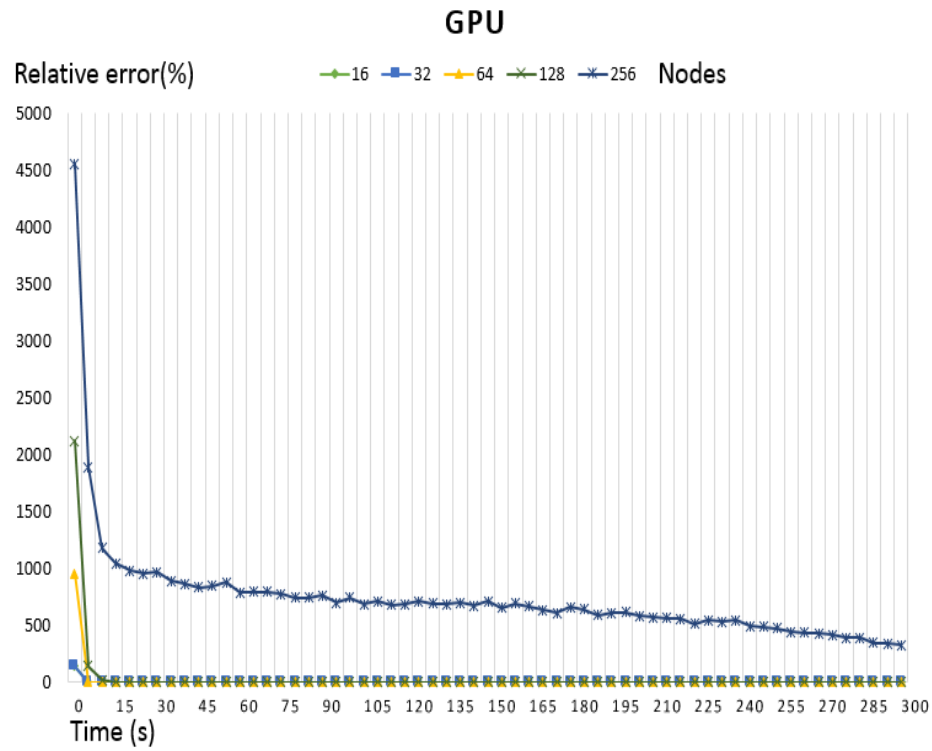
6.2 不同節點數量下的收斂速度

在這一個實驗我們主要要比較經過島模型以及模擬退火法優化後的基因演算法是不是有著比原先有著更優良的收斂率，所以本實驗利用了相同的疊代數量以及相同的時間內做出優化前以及優化後的收斂率比較。首先第一個實驗我們利用時間為基準，我們將時間設定為 5 分鐘，並且觀察在這 5 分鐘內優化前以及優化後的基因演算法其收斂的速度。



圖十四 CPU 在五分鐘內每個 node 數量收斂的相對誤差率

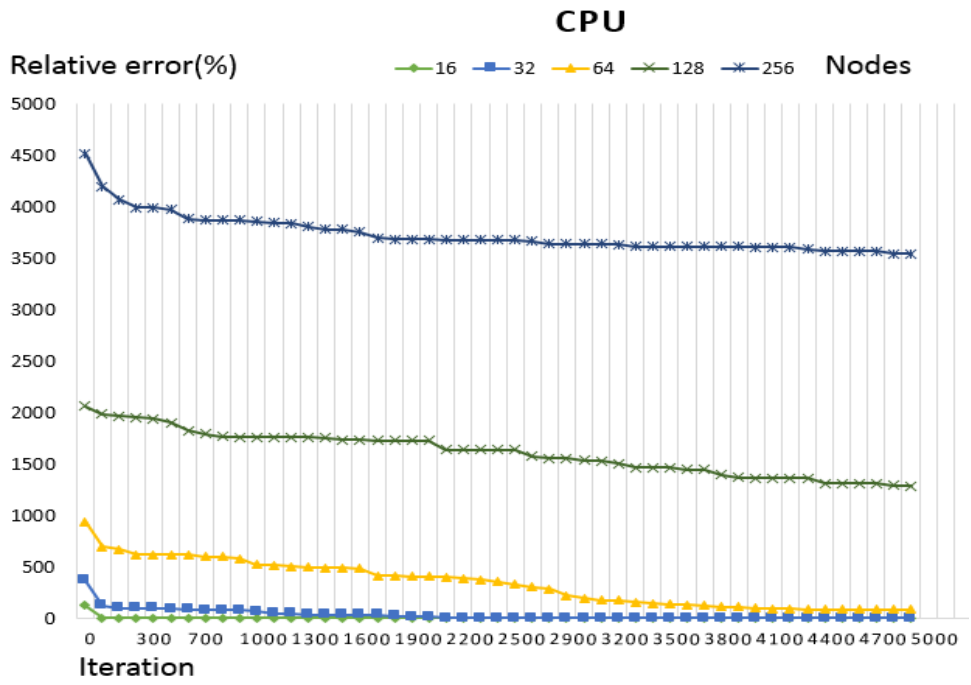
圖十四為優化前並且利用 CPU 進行運算的基因演算法，可以發現其收斂的速度當超過 64 個節點後便無法在 5 分鐘內收斂至最優解，並且隨著節點的增加其收斂的速度成指數遞減。



圖十五 GPU 在五分鐘內每個 node 數量收斂的相對誤差率

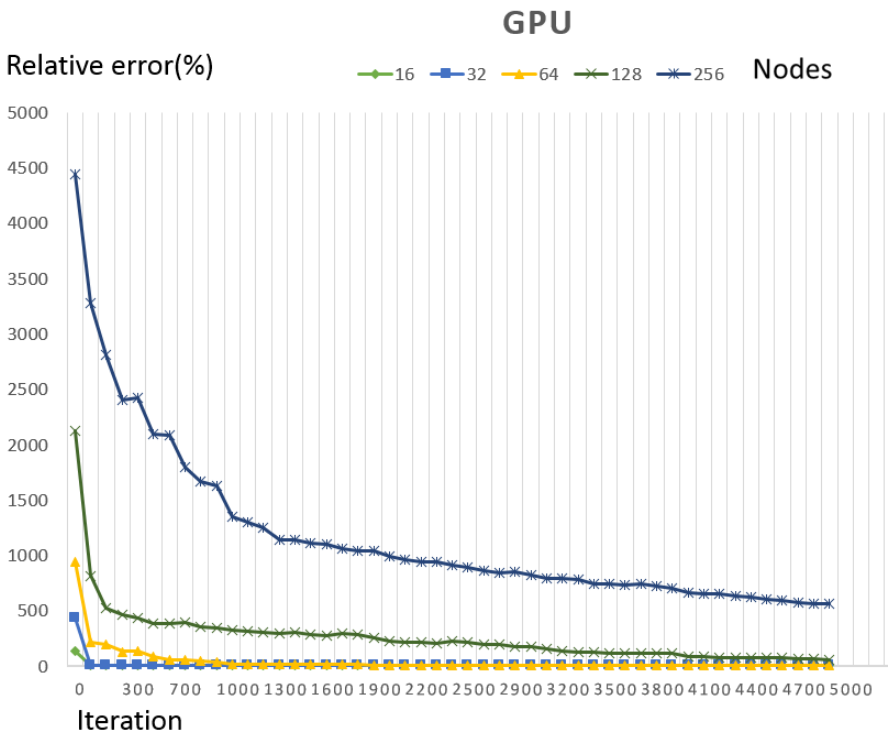
圖十五為利用了 GPU 並且在將基因演算法進行改良後在 5 分鐘內的收斂率，可以發現雖然當節點數到達 256 後便只能找到近似解，但其收斂速度比未優化並且只用 CPU 跑的基因演算法還要來的快非常多。不過從這個實驗仍然無法證明這樣的收斂速度是單純因為 GPU 的高速運算造成的，還是配上優化後的基因演算法才有這樣的

收斂速度。所以第二項實驗我們利用了相同的疊代次數來證明。第二項實驗我們以相同的疊代數量來進行收斂速度上的比較，此次的實驗我們設定為疊代 5000 次為終止條件。



圖十六 CPU 疊代 5000 個世代的每個 node 數量的相對誤差率

由圖十六可以看到未優化的基因演算法並且利用單線程的 CPU 進行運算其收斂速度在此實驗仍是非常的緩慢。



圖十七 GPU 疊代 5000 個世代的每個 node 數量的相對誤差率

由圖十七我們可以看到優化後的基因演算法在同樣的疊代次數上，其收斂速度仍然快於未優化的基因演算法，並且此實驗將 GPU 以及 CPU 的運算速度忽視了，所以可以證明優化後的基因演算法其收斂速度是比未優化的基因演算法還要優秀的。

第七章 總結

根據本篇的實驗結果，我們可以證明基因演算法除了可以平行化之外，也可以在 SIMD 的架構上進行高度的平行化運算。但為了讓基因演算法在 SIMD 的架構上進行高度的平行化運算，我們也更改了不少式子以及其在記憶體存取上的方式，並且我們仍然將過去用於 CPU 上的平行化基因演算法的模型結合進來讓 SIMD 架構上所運行的基因演算法有著更高的收斂速度。而這樣的模型利用了 NVIDIA CUDA 的硬體架構-區塊來實現，其成效讓此演算法加速了 300 倍以上。除了結合島模型來優化之外，我們還結合了模擬退火法，並且經過大量的實驗之後我們發現到了，模擬退火法除了可以當成基因演算法的終止條件之外還可以利用適當的參數讓基因演算法在掉進區域最佳解時有著更快的速度可以跳出來。因此我們將突變的部分利用了模擬退火法進行了取代，而在最後我們將球化退火法的部分利用了雙層的基因演算法來找出其對於目前所要進

行的隨機搜尋的搜尋域的較優參數為何。雖然在本篇並沒有將其實作出來，但這樣的想法也是利用我們所想出的球化退火公式所做的一個延伸，所以在未來我們可以利用這樣的公式結合雙層的基因演算法運算來幫助我們更有效率的找到 NP-hard 問題的最優解。

參考文獻

[1] Whitley, Darrel, Soraya Rana and Robert B. Heckendorn, ”The island model genetic algorithm: On separability, population size and convergence,” Journal of computing and Information Technology,vol. 7, pp.33-48, 1999.

[2] Lindholm, E. Nickolls, J. Oberman, S.,Montrym, J.,”Tesla:A unified graphics and computing architecture,” IEEE Micro, vol. 28,pp. 39-55, 2008.

[3] Toolkit,C.U.D.A.”4.2:CURAND Guide.”, NVIDIA Corporation, Santa Clara, 2012.

[4] Pit, Laurens Jan.”Parallel genetic algorithms,” MS(Computer Sci.), 1995.

[5] Aarts, Emile, and Jan Korst.”Simulated annealing and Boltzmann machines,” 1988.

[6] Lenstra, Jan Karel, AHG Rinnooy Kan, and David B. Shmoys.”The traveling salesman problem:a guided tour of combinatorial optimization,” New York: Wiley, vol. 3, 1985.

[7] Han, Tianyi David, and Tarek S. Abdelrahman, ”hiCUDA: High-level GPGPU programming,” IEEE Transactions, vol 22.1, pp. 78-90, 2011.

[8] Nouredine Melab, El-Ghazali Talbi, “GPU-based island model for evolutionary algorithms,” Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, New York, USA: ACM Press, pp.1089-1096, 2010.

[9] Maxim Vedenev; http://simulations.narod.ru/

[10] MUNSHI, Aaftab, “ The opencl specification,“ *2009 IEEE Hot Chips 21 Symposium (HCS)*. IEEE, pp. 1-314, 2009.

[11] Karimi, Kamran, Neil G. Dickson, and Firas Hamze, "A performance comparison of CUDA and OpenCL," *arXiv preprint arXiv:1005.2581*, 2010.

[12] Robilliard, Denis, Virginie Marion, and Cyril Fonlupt, "High performance genetic programming on GPU," ACM *Proceedings of the 2009 workshop on Bio-inspired algorithms for distributed systems*, pp.85-94, 2009.

[13] Yu, Qizhi, Chongcheng Chen, and Zhigeng Pan, "Parallel genetic algorithms on programmable graphics hardware," *International Conference on Natural Computation*. Springer Berlin Heidelberg, pp. 1051-1059, 2005.

[14] Langdon, William B, "A many threaded CUDA interpreter for genetic programming," *European Conference on Genetic Programming*. Springer Berlin Heidelberg, pp. 146-158, 2010.

[15] Reinelt, Gerhard, "TSPLIB—A traveling salesman problem library," *ORSA journal on computing,* vol 3.4, pp. 376-384, 1991.

[16] Dantzig, George, Ray Fulkerson, and Selmer Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America,* vol 2.4, pp.393-410, 1954.

[17] Pardalos, P. M., Pitsoulis, L., Mavridou, T., & Resende, M. G, "Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing, tabu search and GRASP,"*International Workshop on Parallel Algorithms for Irregularly Structured Problems*. Springer Berlin Heidelberg, pp. 317-331, 1995.

[18] Wong, Kit Po, and Yin Wa Wong, "Genetic and genetic/simulated-annealing approaches to economic dispatch," *IEE Proceedings-Generation, Transmission and Distribution,* vol 141.5, pp. 507-513, 1994.

[19] Hwang, Chii-Ruey, "Simulated annealing: theory and applications," *Acta Applicandae Mathematicae,* vol 12.1, pp.108-111, 1988.

[20] Mazza, Christian, "Parallel simulated annealing," *Random Structures & Algorithms,* vol 3.2, pp. 139-148, 1992.