Parallelizing Genetic Algorithm

李鎮宇 0756615，陳穎劭 0756052，張家銘 0756099

Code: https://github.com/wcps0401/parallel_programing_final

Abstract —基因演算法(GA)是一個解決最佳化問題的工具，它是人類依照生物學中「適者生存，不適者淘汰」的觀念所發展出來的一種演算法利用「選擇、複製」、「交配」、「突變」等步驟去尋找最適合環境的基因，然而隨著族群數量的龐大，每一代間的計算速度也越來越慢。因此，本專題利用 pthread, OpenMP, CUDA 等平行加速技巧，希望能讓基因演算法可以在一代間需要完成的動作時間縮短，像是交配、變異、算 fitness 等…，以達到更快速的找到優化的目標答案。

I. INTRODUCTION

基因演算法(Genetic Algorithm)是一種參考世界上生物界適者生存的機制而成的演算法，是一種特化隨機的搜索方法。在 1975 年由美國的約翰·霍蘭德（John Holland）教授和他的同事在研究細胞自動機時率先提出，後來約翰·霍蘭德也因為這個演算法被稱為遺傳演算法之父。而這樣的演算法在早期還僅僅只限於理論方面，實際應用上的效果並不多，直到第一屆世界遺傳演算法大會，隨著科技的日新月異，電腦的計算能力與應用需求變高，基因演算法逐漸進入到可以實際應用的階段。在這之後，各式各樣不同種類的基因演算法被應用在資料分析、時間安排、預算、未來趨勢預測等等各領域的組合最佳化問題。

而基因演算法主要是運用進化生物學中的遺傳現象、突變現象、天擇以及雜交來做發展。基因演算法通常用以解決一些最佳化的問題。對於每個最佳化難題，基因演算法會有一定數量的候選個體解，將這些個體解視作染色體，運用自然界的各種遺傳現象來對這些染色體做變化，進而使得整個群體邁向更好的方向進化發展。在最初的親代會是完全隨機的個體，以這些隨機的個體開始做演化，一代一代的產生子代，而在每一代中會評估整個群體的適應值，並將好的群體被選為親代交配的機率加大，讓優良的基因得以有更高的機率傳遞他的基因給下一代，在設定的繁衍代數中不斷的演化。可參考 Fig.1。

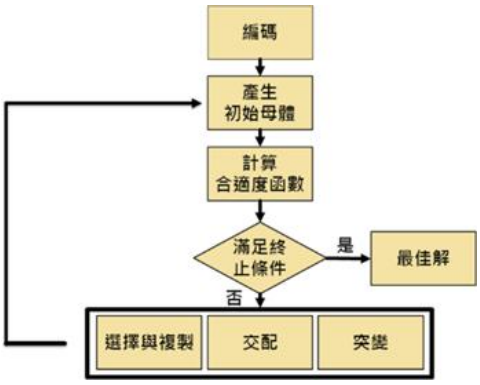

Fig.1 一般基因演算法流程圖

在基因演算法中，待解決的最佳化問題解被稱為個體，可以用變數序列來做表示，稱做染色體或基因串。將最佳化問題解轉換成染色體或基因串的過程稱為編碼。編碼完成之後隨機生成一定數量的個體做為親帶，有時也可依據題目的關係操作這個隨機

的過程，讓產生的基因更加接近解，提高初代物種的適應值。接著可以運用這些親代來做繁殖的動作。繁殖的過程通常會包括交配(Crossover Fig.2)與變異(mutation Fig.3)。在交配之前會先計算當前群體的適應值，用來表示這些不同的解對於最佳化問題的解決程度，分數越高代表擁有越有可能會是問題的解，在適應值的計算設置會是非常重要的。在交配的動作中演算法會以母代的適應值做為參考來選擇，會讓適應值較高的個體擁有較高的交配權，有較高的機率會被選上，相反的適應值較低的個體則只會有較低的機率被選上。在每代中會選擇一定數量的個體來做交配，當每次交配發生時通常會互相交換兩個親代之間的染色體，交換的方式也非常多種可以選擇，可以視題目的狀況決定交換方式，通常會視將母代的兩個染色體隨機找一個點位做切割，第一個子代會是前半段父代加上後半段母代，而第二個子代則會是前半段母代加上後半段父代，與前一個子代相反。交換完成之後即可產生新的子代，做為下一代繁殖的母代群體。而在完成交配之後是突變，在突變的過程中會選擇子代病設定突變的機率，突變的機率通常並不會太大，當突變發生時則是會替換掉當前突變位子內的數值，如果是二進位的表示的話則會是 1 變 0 或 0 變 1。如果是字元則會替換當前字元為隨機字元。在經過非常多次演化的過程（評分適應值、選擇、交配、突變），由於會在每次選擇的過程中留下適應度好的個體，使得適應度差的個體被篩選掉，將會一代一代增進整體群體的適應度發展方向。在不斷的重複這樣的一個過程時即可產生非常接近最佳解的解答。

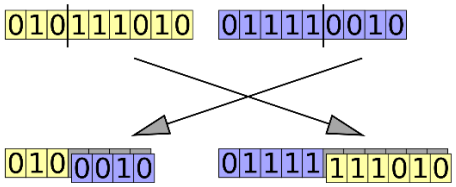

Fig.2 交配(crossover)

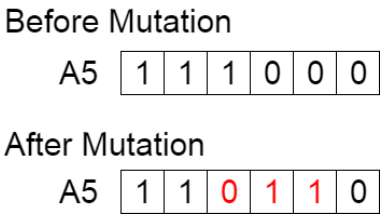

Fig.3 突變(mutation)

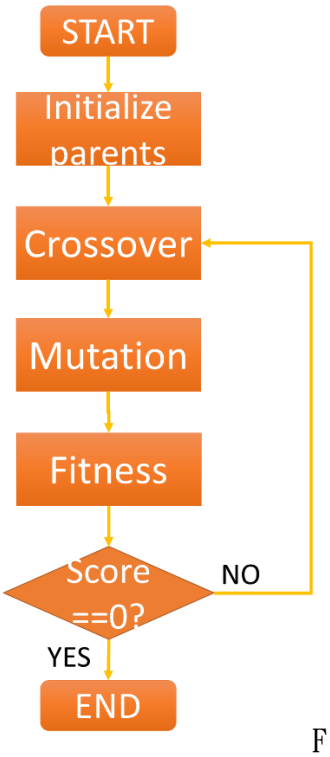

Fig.4 流程圖

II. PROPOSED SOLUTION

我們這次的程式主要是運用基因演算法來做尋找字串的動作。會選擇字串尋找作為題目主要是可以用字串的演變清楚地看出基因演算法本身的特性，也就是會逐漸繁衍出優良的世代，而我們認為平行化的方法可以簡

單方便的套用在其他不同種最佳化題目的基因演算法上面。

本程式主要步驟分為三個部分，如下 Fig.5 所示，整體來說每次的世代內會做 crossover、mutation 與計算 fitness 的動作。經過設定的繁衍代數上限或是已經找到答案之後即會結束基因演算法的搜尋，並給出適應值最高的個體做為最佳解。整體流程圖可見上面 Fig.4。以下將會詳盡敘述個個步驟所做的方法以及平行化的部分。

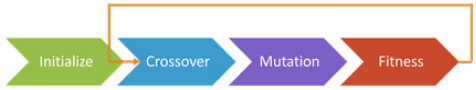

Fig.5 程式簡易流程

	Initialize	Crossover	Mutation	Fitness
Time (s)	0.155 s	15.894s	21.066s	11.35s
Time (%)	0.319 %	32.795%	43.467%	23.418%

表 1 各步驟時間占比表

A. 初始化親代

在本程式中最一開始要由基因池中隨機挑選字元組合產生母代的群體集合(Fig.6)，之後的步驟將會由這些產生的群體開始進行挑選演化。在這個步驟中並不會耗費太多的時間(表 1)，故沒有在這一步驟施行平行化。

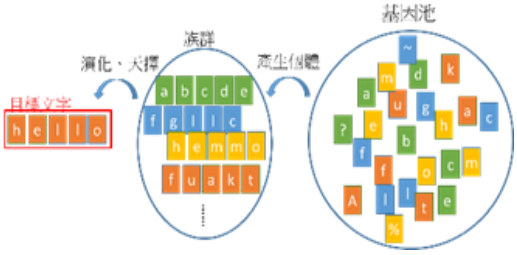

Fig.6 初始化示意圖

B. Crossover 交配

在交配之前會先做選擇(Fig.7)的動作，選擇兩個親代做交配，而本程式選擇的方法為在選擇每一個父代或母代時，由目前整體群體中挑選兩個目標，由適應值較高者可候選為父母代。透過這樣的做法可以很簡單的實現出自然界基因較好的個體有較高的機率被選作親代的特性，實現出適者生存的原則。在父母代挑選完成之後就會做 crossover 的動作。在這邊我們使用的演算法為 1-point crossover(Fig.8)，隨機選定一個位置作為切點，將切點後的字串互相交換藉此產生新的子代。在這個步驟中因為每一代都會執行 crossover 一次，顧 crossover 占整體程式執行時間約 32%(表 1)，故我們有對 crossover 進行平行化。

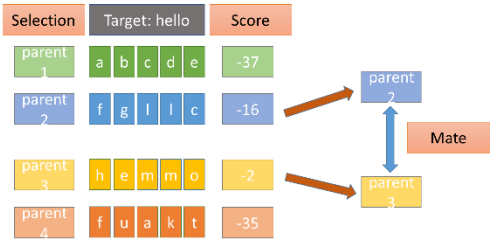

Fig.7 selection 選擇

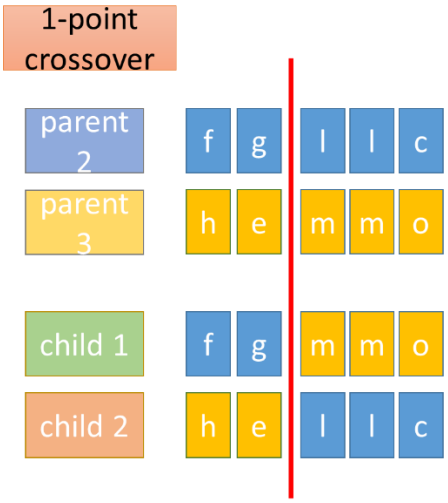

Fig.8 corssover 交配

C. Mutation 突變

在完成 crossover 之後會讓產生的子代進入 mutation 中進行突變(Fig.9)。這邊會將全部的子代內的字元一個一個去做突變，突變的機率為百分之一，每個字元都有百分之一的機率去做突變，當突變發生時將會將該位置的字元替換成隨機的一個字元。在這裡的步驟中由於會將所有子代的每個字元去做 mutation 機率判斷，程式需要掃過全部的字元，故會占整體程式很大的一部分時間，約占 43%的執行時間(表 1)。故也有對此函數做平行化的優化。

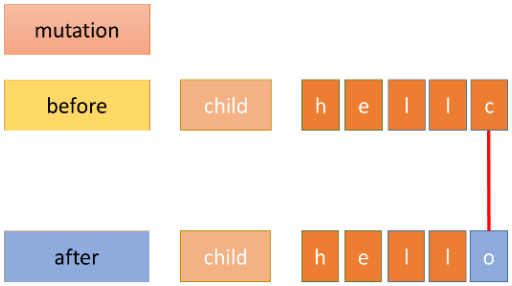

Fig.9 mutation 突變

D. Fitness 適應值計算

在完成 crossover 與 mutation 的步驟之後就會對每個新產生的子代做 fitness 適應值的計算，計算方法為檢查每個子代內的每個位置字元與目標答案字元的 ascii code 差異絕對值相加再給予一個負值。0 表示完全沒有錯誤，找到正確答案，數字越小表示離正確的答案距離越遠。這個步驟占用整體程式執行時間約 23%(表 1)，亦是可以平行化的部分。

如上所示，由於 crossover、mutation 與 fitness 的部分會在每一代的繁衍不斷地被執行，占用整體的執行時間，故我們可以對這些部分做平行化。這邊我們分別使用的平行化方法分別為 pthread、openMP 與 CUDA。

在 pthread 平行化的部分，我們會在每代繁衍的過程中分各個 thread 做平行化。會將整體需要產生的 children 個數平分到每個 thread 內，每個 thread 內各自執行 crossover、mutation 與 fitness 產生差不多數量的 children，互相並不會干擾。假設 pool 數量(親代數量)是 20000 的狀況下欲產生 20000 個子代，在 thread 數量為 4 的狀況下每個 thread 會被分配到產生 5000 個子代。而因為只會對親代去做讀取的動作，並不會去修改到親代的內容，故不需要 mutex 讓 thread 之間互相等待。

而在 openMP 的部分，實際平行化概念方式與 pthread 大同小異，只是在操作上更為便利，不需要使用者

自己分配。

而在 CUDA 平行化的部分因為每一次的 crossover 是隨機選出 2 個 parent 產生出 4 個 child，而在 pool 數是 20000 的情況下需要 5000 對父母，因此我們設了 10 個 block 每個 block 有 500 個 threads，所以共有 5000 個 threads，每一個 threads 會同時進行 crossover、mutation、fitness，而 CPU 只需傳一次最初代的 parent 給 GPU，接下來就是在 GPU 進行所需的迭代後再傳回 CPU，而這裡是用"cudaMemcpyDeviceToDevice"的方法將這代的 child 給值到下一代的 parent。

III.EXPERIMENTAL METHODOLOGY

本次的實驗我們使用了課程所提供的平台，處理器為 Intel(R) Core(TM) i7 CPU @2.93GH，為四核心八執行緒，顯示卡為 NVidia 1060，作業系統是 Ubuntu 16.04，記憶體大小為 8GB。

實驗方式為設定一組目標字串，測量程式的執行時間。字串長度有短、中、長不同長度，分別為 12、60、120 個字元，去測試不同平行化方式的執行速度。實驗分為兩部份，分別以不同的平行化方式實驗。第一部分為 pthread 和 openmp，分別測試 pthread 和 openmp 在 1 thread、2 threads、4 threads、8 threads；第二部分為 cuda，測試 cuda 在 5000 threads 下的執行時間。

由於程式是由隨機演化產生字串，因此找到目標字串的代數並不一定，為了公平且方便直觀的測量程式的執行時間，不同的平行化方式實驗統一執行到 10000 代，紀錄程式到 10000 代的執行時間。因為每次測試時間皆會有所浮動，因此實驗對所有的平行化方式與不同長度目標字串接執行 5 次，取平均作為結果。

IV. EXPERIMENTAL RESULTS

第一部份，實驗在 pthreads 和 openmp 在不同字串長度(12、60、120 個字元)和不同執行緒數(1、2、4、8 threads)下的執行結果。在字串長 12 時(Fig.10)可看出在 threads 數為 1、2、4 時，pthreads 和 openmp 程式行時間皆是遞減的。需注意的是，openmp 在 thread 數為 1 時的執行時間較 serial 和 pthread 多了 20 多秒，這是因為 openmp 程式中使用了 clock()做為產生 random seed 的方式而 serial 和 pthread 皆使用 time()，因此，openmp 程式較其他兩者花較多的時間，但依然可以看到 pthread 和 openmp 皆是有效率的平行程式，減少執行時間。

在 8 threads 時，可以觀察到 pthread 程式執行時間反而較在 4 threads 時的執行時間久，而 openmp 在 8 threads 上效能也並沒有預期的好，並沒有加速至 4 threads 執行速度的近兩倍，這是因為實驗平台 cpu 支援 intel hyper-threading 技術，超執行緒技術是在 cpu 上增加邏輯運算單元，讓一個 cpu 可執行兩個 threads，然而兩個 threads 依然共用相同的整數運算單元。在本程式中，隨機的 crossover、mutation 和計算 fitness 皆需要大量的整數運算單元，因此在 pthread 上反而造成

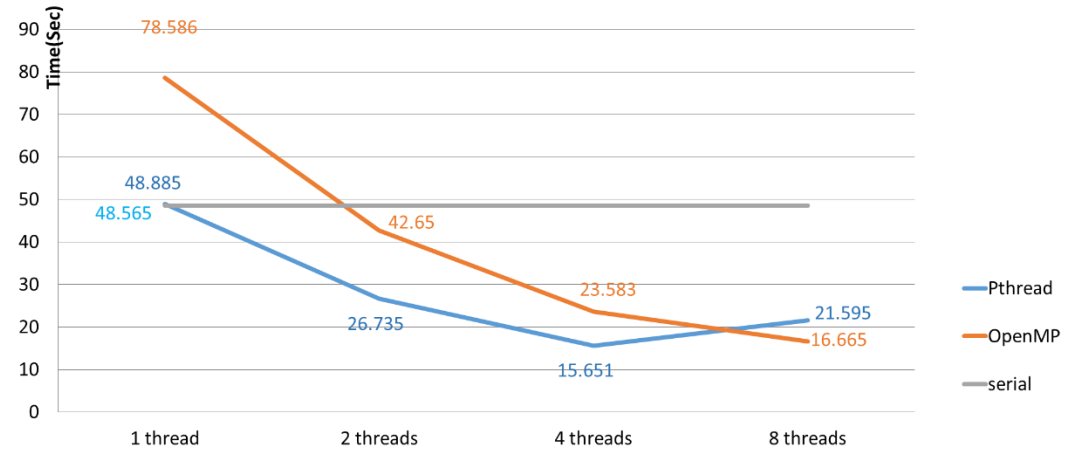

Fig.10 pthread 和 openmp 在字串長 12 下執行 10000 代的時間

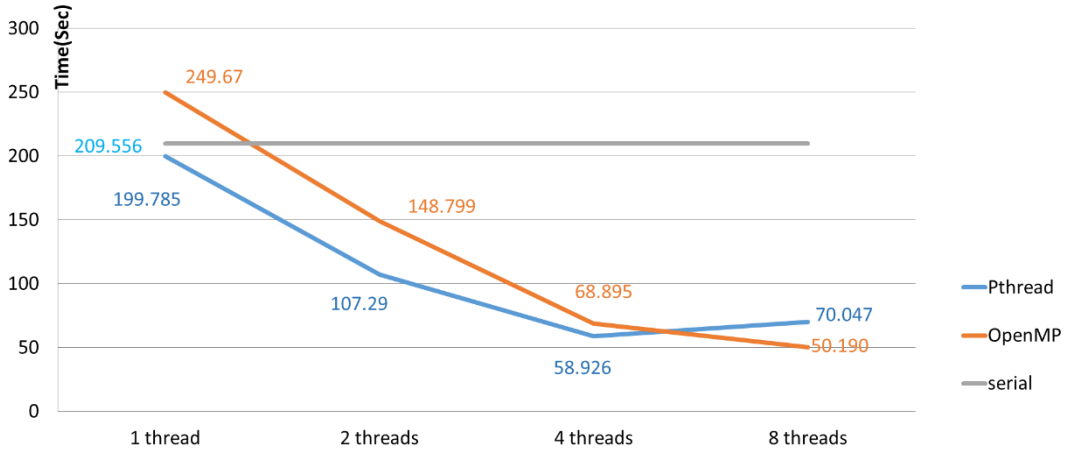

Fig.11 pthread 和 openmp 在字串長 60 下執行 10000 代的時間

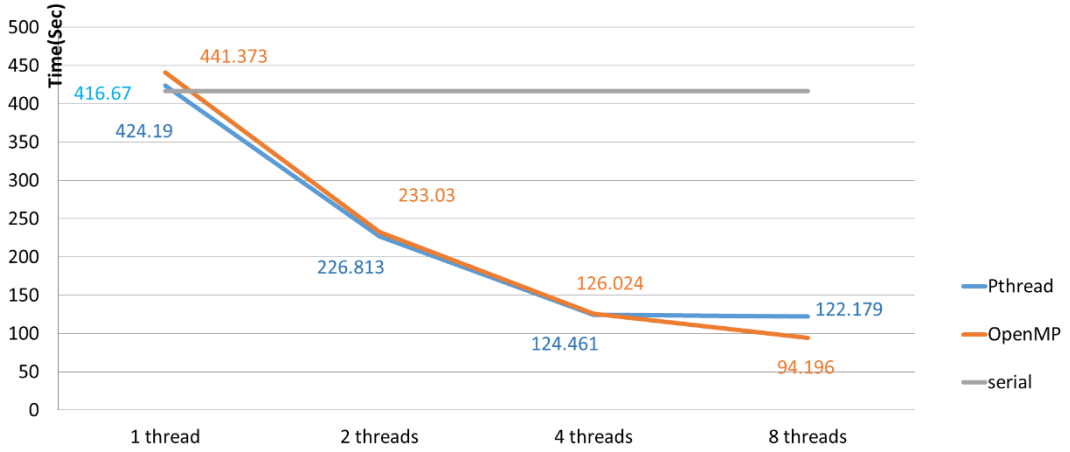

Fig.12 pthread 和 openmp 在字串長 120 下執行 10000 代的時間

overhead，增加了 8 threads 的執行時間。而 openmp 8 threads 較 4 threads 有些微加速，並沒有出現 pthread 執行時間不減反增的情形，應是 openmp 對 hyper-threading 有較佳的優化，可以有效利用其中多增加的邏輯運算單元。在字串長 60 與 120 時(Fig.11、12)，可以看到實驗結果與字串長度 12 時相似，openmp 因 random seed 的取法不同而造成 thread 時執行時間較 serial 和 pthread 久。Hyper-threading 造成

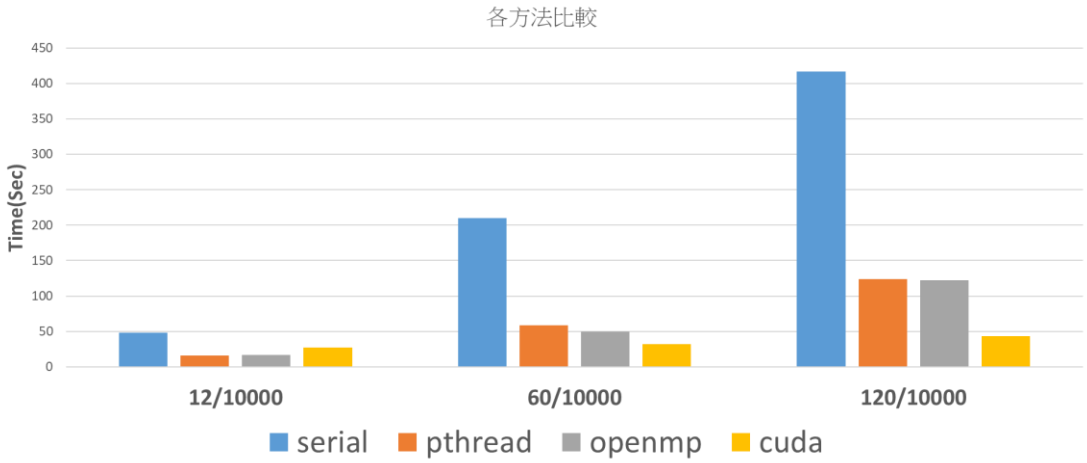

Fig.13 各方法在不同字串長下執行時間比較

的 8 threads 減速問題依然可以在圖中觀察到。

第二部份，實驗 cuda 在 5000 threads 下的執行時間。可以看到(Fig. 13)除了 12 字元的字串外，cuda 的執行時間皆小於 pthread 和 openmp。這是因為 cuda 程式在一代之間做平行，在代跟代之間需要重新讀取資料，而字串長度 12 的一代執行時間很短，造成讀取時間 overhead。而在字串較長較複雜的情況，一代的執行時間長，overhead 時間的比例降低，因此加速較為顯著。

最後，統整 pthread、openmp 和 cuda 的加速倍率，pthread 和 openmp 取未被 hyper-threading 技術影響的 4 threads 做比較(Fig. 14、15、16)。可以觀察到 pthread 和 openmp 在 4 threads 的加速可達到 3 點多倍，這是因為程式在每一代之中做平行化，但每一代的計算時間很短，所以會導致平行化的 overhead 變高，使得效率無法接近 4 倍。而在字串比較複雜一些的情況下，因為較長的字串在每一個代裡面會花費較多的時

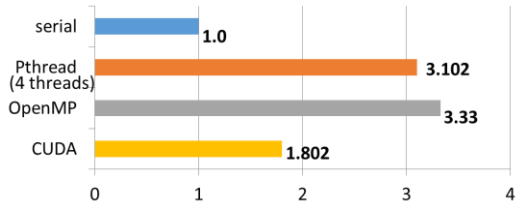

Fig.14 字串長度 12 時加速倍率

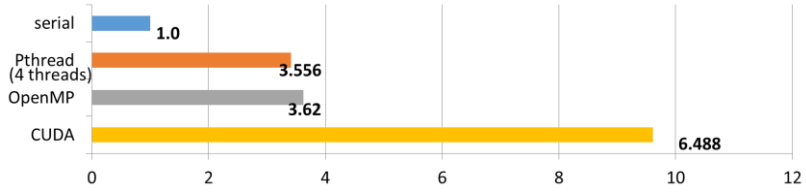

Fig.15 字串長度 60 時加速倍率

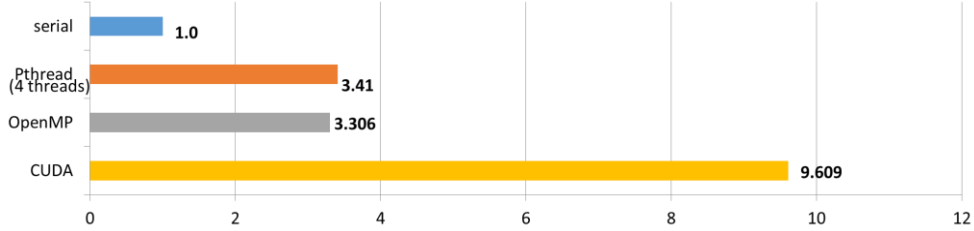

Fig.16 字串長度 120 時加速倍率

間，可以讓 overhead 的比例降低，可以使得平行化的效率變高。

V. RELATED WORK

1. 遺傳程序

遺傳程序是 John Koza 與遺傳算法相關的一個技術，在遺傳程序中，並不是參數優化，而是電腦程式優化。遺傳程序一般採用樹型結構表示電腦程式用於進化，而不是遺傳算法中的列表或者數組。一般來說，遺傳程序比遺傳算法慢，但同時也可以解決一些遺傳算法解決不了的問題。

2. 互動式遺傳演算法

互動式遺傳演算法是利用人工評價進行操作的遺傳演算法，一般用於適應度函式無法得到的情況，例如，對於圖像、音樂、藝術的設計和「最佳化」，或者對運動員的訓練等。互動式遺傳演算法將傳統的進化機制與人工智能評價相互結合，可以有效的解決性能指標難以用精確函數表示的優化問題。

3. 模擬退火法(Simulated Annealing)

模擬退火是解決全局最佳化問題的另一個可能選擇。它是通過一個解在搜尋空間的隨機變動尋找最佳點的方法：如果某一階段的隨機變動增加適應度，則總是被接受，而降低適應度的隨機變動根據一定的機率被有選擇的接受。這個機率由當時的退火溫度和適應度惡化的程度決定，而退火溫度按一定速度降低。從模擬退火演算法看，最佳化問題的解是通過尋找最小能量點找到的，而不是尋找最佳適應點找到的。模擬退火法也可以用在標準遺傳演算法裡面，只要把突變率隨時間逐漸降低就可以。

4. 禁忌搜索演算法

禁忌搜索（Taboo Search，簡稱 TS）的思想最早由 Fred Glover(美國工程院院士，科羅拉多大學教授)提出，它是對局部領域搜索的一種擴展，是一種全局逐步尋優演算法，是對人類智力過程的一種模擬。TS 演算法通過引入一個靈活的存儲結構和相應的禁忌準則來避免迂迴搜索，並通過藐視準則來赦免一些被禁忌的優良狀態，進而保證多樣化的有效探索以最終實現全局優化。相對於模擬退火和遺傳演算法，TS 是又一種搜索特點不同的 meta-heuristic 演算法。迄今為止，TS 演算法在組合優化、生產調度、機器學習、電路設計和神經網路等領域取得了很大的成功，近年來又在函數全局優化方面得到較多的研究，並大有發展的趨勢。本章將主要介紹禁忌搜索的優化流程、原理、演算法收斂理論與實現技術等內容。

VI. CONCLUSION

在經過上面的實驗後，對於各種長度的字串，3 種方法的加速表現也不一樣，可能在短字串加速效果最好但在長字串就沒有那麼好了，經過歸納與分析，我們整理出以下幾點結論。

A. pthread

在短字串的表現上是三種方法中最優的，而又以 4 threads 為最快，每條 thread 的平均使用率為最高，在 8 threads 反而會變慢是因為測試環境是 intel hyper-threading 的關係，但 pthread 在字串長度為 120 字元也就是長字串時，表現就會輸其他兩種方法了。

B. openmp

openmp 為高階平行化函式庫，許多功能都是被包裝好的，在複雜的程式中，很難針對個案去做特殊的設定，所以實驗後發現它在 4threads 時，短字串及中字串的實驗結果都是略輸 pthread，但是它在 8threads 時卻可以繼續加速，原因是 openmp 有做動態規劃的優化讓某些 thread 不會因為先做完而閒置，讓每個 thread 皆可被利用，進而提升執行速度。而 openmp 在長字串時的加速結果略快於 pthread。

C. cuda

cuda 因為是在 GPU 做加速，所以 thread 數量遠大於 pthread、openmp，但是他的前置動作必須先將 CPU 上面的 data 傳到 GPU 的 memory，而等到做完後還要再傳回 CPU，這會花掉了一段時間，導致它在短字串時加速效果是低於 pthread、openmp 的，但在長字串就優於前面兩種方法了，字串越長，加速倍率越高。

D. 選擇平行化工具

因為在不同的長度時，表現最好的方法都不盡相同，因此，我們必須因應題目的情況，使用不同的平行化工具，以便達到最大的效率。

E. GA 演算法是可被平行的

雖然在每一個 iteration 間是有遺傳關係不可被平行的，但是我們在同一帶間對它的方法做平行，因為同代間個體的交配、變異、算分是可被平行的，實驗後發現整體平行化效果是顯著的。

VII. REFERENCES

- Global Optimization Algorithms – Theory and Application – http://www.it-weise.de/projects/book.pdf
- XingYu Fu, JinHong Du , YiFeng Guo , MingWen Liu, Tao Dong , XiuWen Duan – A Machine Learning Framework for Stock Selection https://arxiv.org/pdf/1806.01743.pdf
- MBAlib 遺傳演算法 https://wiki.mbalib.com/zh-tw/%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95
- Goldberg, David E (1989), 遺傳演算法：搜尋、最佳化和機器學習，Kluwer Academic Publishers, Boston, MA.
- Poli, R., Langdon, W. B., McPhee, N. F. A Field Guide to Genetic Programming. Lulu.com, freely available

from the internet. 2008. ISBN 978-1-4092-0073-4.

- Schmitt, Lothar M (2001), 遺傳演算法理論，Theoretical Computer Science (259), pp. 1-61
- Schmitt, Lothar M (2004), 遺傳演算法理論（二），Theoretical Computer Science (310), pp. 181-231