Parallelize Genetic Algorithm using Thrust

蔡佩珊
0456024
National Chiao Tung University
1001 University Road, Hsinchu, Taiwan 300, ROC
pstsai@nclab.tw

黃奕齊
0556032
National Chiao Tung University
1001 University Road, Hsinchu, Taiwan 300, ROC
you74674@gmail.com

ABSTRACT

我們建構了一個 C++ template Class GeneticAlgorithm，其中所採用的篩選 (Selection)、交配 (Crossover) 及突變 (Mutation) 方法是最普遍且通用的演算法，因此使用上有很大的彈性，可以應用於各式各樣的問題。
在使用 Class GeneticAlgorithm 求解時，使用者無須了解基因演算法的步驟細節，僅需建構與自身目標問題相關的類別：指定初始化、適應度計算及適應度比較的方法，繼承專案內的 Class Problem 後，即可套用。
本專案以 Thrust Libary 實作，因此支援多種平行化方法，例如：OpenMP、TBB 以及 CUDA，大幅改善了基因演算法收斂耗時的卻點，使其可以應用在更複雜的問題。

CCS Concepts

• **Computing methodologies** → **Parallel algorithms; Genetic algorithms;**

Keywords

Genetic Algorithms; Parallel Computing; Thrust Library

1. INTRODUCTION AND MOTIVATION

基因演算法 (Genetic Algorithm) 是解最佳化問題的演算法，屬於演化計算 (Evolutionary Computation) 的一個分支。此法藉由模擬生物演化的機制：將最佳化問題類比為自然環境，在此環境下存在一個群體 (Population)，群體內有多個候選解，稱為個體 (Individual)，每個候選解內容可表示成變量序列，稱為染色體 (Chromosome)。進化從隨機的初始群體開始，在每個世代評價個體的適應度 (Fitness)，適應度高的個體通過天擇 (Selection) 可以進行交配 (Crossover) 及突變 (Mutation)，遺傳生成下個世代的個體，周而復始，直到收斂至全域最佳解、局部最佳解或者滿足終止條件。詳見 Figure 1。
基因演算法的流程跟目標問題沒有太大的相關，因此很容易使用，常被用來解最佳化問題 (Optimization Problem) 以及搜索問題 (Search Problem)。但是當個體數量變多、染色體維度變大、適應度計算變複雜的時候，求解所需的收斂時間便會大幅增加。因此，如何加速基因演算法是個值得重視的議題，而平行計算 (Parallel Computing) 便是個最能保證提升效能的方法。

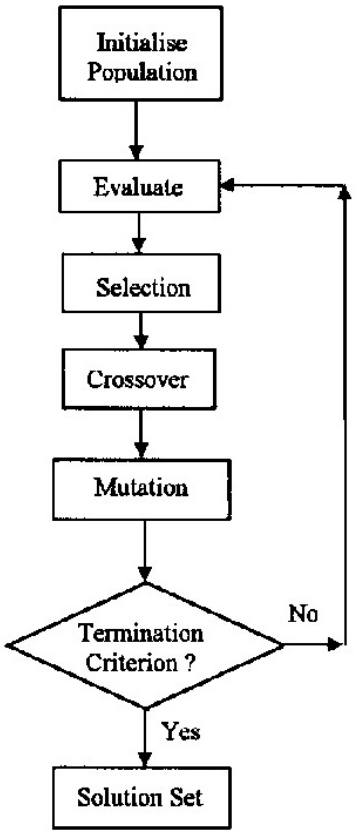

Figure 1: 基因演算法的流程圖

2. PROPOSED APPROACHES

2.1 Parallel Model

在基因演算法中，個體即為解，因此許多的運算操作都是以個體為單位進行。我們依照此特性所採取的平行化單位即是個體。以下為平行化細節，其中 N_I 為個體數，N_T 為線程 (Thread) 數。

2.1.1 Initialization

ACM ISBN .
DOI:

個體的初始化是互相獨立且計算量相當，因此我們把個體均分至線程進行初始化。
在本專的平行化架構下，每個線程：初始化 N_I/N_T 個個體。

2.1.2 Evaluation

個體的適應度計算是互相獨立且計算量相當，因此我們把個體均分至線程進行適應度計算。
在本專的平行化架構下，每個線程：計算 N_I/N_T 個個體的適應度。

2.1.3 Selection, Crossover and Mutation

演化的過程有三個步驟，篩選、交配及突變，雖然篩選及突變是以單個個體為單位。但是考慮到每個步驟的計算量不大，以及記憶體配置（例如，在篩選的時候以個單個個體為單位進行，因為後面的交配步驟是以兩個個體為單位進行，為了能讓線程間能互相存取篩選出來的結果，勢必得將資料配置在線程的生命週期之外，但是這個資料除了這個操作外即無用途，造成記憶體資源浪費）、線程啟動的代價（例如，在大部分的應用中，突變的機率趨近於零，為了那少量的突變操作，而要在每次演化都特地開線程會增加不必要的成本）等問題。因此我們將三個步驟合併，以交配的兩個個體為單位進行。
在本專的平行化架構下，每個線程：進行 $N_I/2 \times N_T$ 次，篩選出兩個父代、將兩個父代進行交配產生兩個子代、將兩個子代進行突變。

3. IMPLEMENTATION

3.1 Thrust Library

我們的實作採用 CUDA 的 Thrust Library，有以下特色：

- 與 STL 相容的 API。
- 支援多種平行方式 (OMP、TBB、CUDA)。
- Thrust 會根據輸入大小自行分配執行緒，不需要自行計算。

3.2 Implementation Details

記憶體配置部分可直接利用 thrust::device_vector，省下 cudaMalloc 等繁雜的程式碼。但是因為 thrust::device_vector 不支援巢狀，所以將所有空間配置好後，還需要將每段空間的指標指定給每個個體。
在基因演算法中，有這幾個種類的迴圈：

- 世代的迴圈：此迴圈無法平行化，因為每個世代都是根據前一個世代的結果來進行。
- 個體的迴圈；使用 thrust::for_each 將個體的迴圈平行化。
- 找尋最佳個體的迴圈：使用 thrust::min_element 直接呼叫平行化的函式。

4. EXPERIMENTAL RESULTS

我們將專案分別應用於連續最佳化問題 Ackley Function 與離散最佳化問題 OneMax 進行測試。

4.1 Ackley Function

4.1.1 Serial version

如 Figure 2. 所示。因為原版和 Thrust 版的 Serail 的時間非常不平滑，所以以下 Speedup 將利用 Contour 圖來表示。
（上）原版的 time per iteration。
（中）Thrust: serial 的 time per iteration。
（下）時間比值，Thrust 版花費時間大致相同，原版稍微快一點。

4.1.2 OMP version

如 Figure 3. 所示。大約達到 3 6 倍的 speedup，但較高的效果主要集中在 gene size 較小或 population 較小的情況，在兩者都較大的情況 speedup 較不明顯。
（上）time per iteration
（下）speedup(OMP/serial 的時間比值)

4.1.3 CUDA version

如 Figure 4. 所示。最多能達到 12 倍的 speedup，主要在 population 足夠大的時候。
（上）time per iteration
（下）speedup(CUDA/serial 的時間比值)

4.1.4 OMP vs. CUDA

如 Figure 5. 所示。cuda 在 gene size 和 population 都過了 3000 左右之後，就都贏過 omp，差距大約為 2 倍。
（上）time per iteration
（下）CUDA/OMP 的時間比值

4.1.5 時間分割圖

如 Figure 6. 所示。sequential 的時間不平滑成長的原因在於 compute fitness 的部分。但很奇妙的 OMP 中卻沒有出現這個問題。雖然 CUDA 在這個問題有最快的速度，但花費了大量的時間在 update global best 上。
（上）serial
（中）OMP
（下）CUDA

4.2 OneMax Function

4.2.1 Serial version

如 Figure 7. 所示。thrust 版在 gene size 小的時候花的時間最多約是原版的 1.2 倍，但在 gene size 大時約為原版的 0.95 倍。
（上）原版與 thrust sequential 的 time per iteration
（下）時間比值。

4.2.2 OMP version

如 Figure 8. 大約達到 3.2 5.3 倍的 speedup。除了 population 和 gene size 都很小的情況以外，幾乎都有 4 倍多的 speedup。
（上）time per iteration
（下）OMP/sequential 的時間比值。

4.2.3 CUDA version

如 Figure 9. 要在 population 和 gene size 都大於 2000 的情形才能有一點點的 speedup(最多約 1.7 倍)，以下時反而比較慢 (0.5 倍左右)。
（上）time per iteration
（下）CUDA/sequential 的時間比值。

4.2.4 OMP vs. CUDA

如 Figure 8. 大約達到 3.2 5.3 倍的 speedup。除了 population 和 gene size 都很小的情況以外，幾乎都有 4 倍多的 speedup。

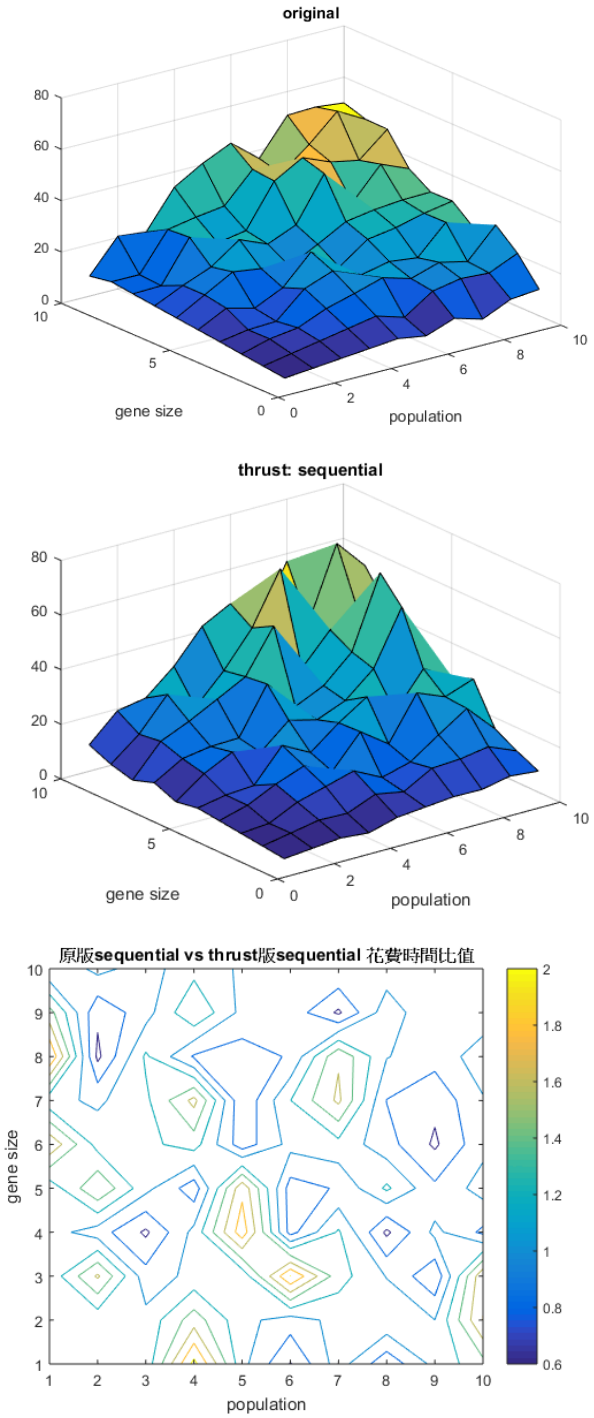

Figure 2: Ackley Function serial version

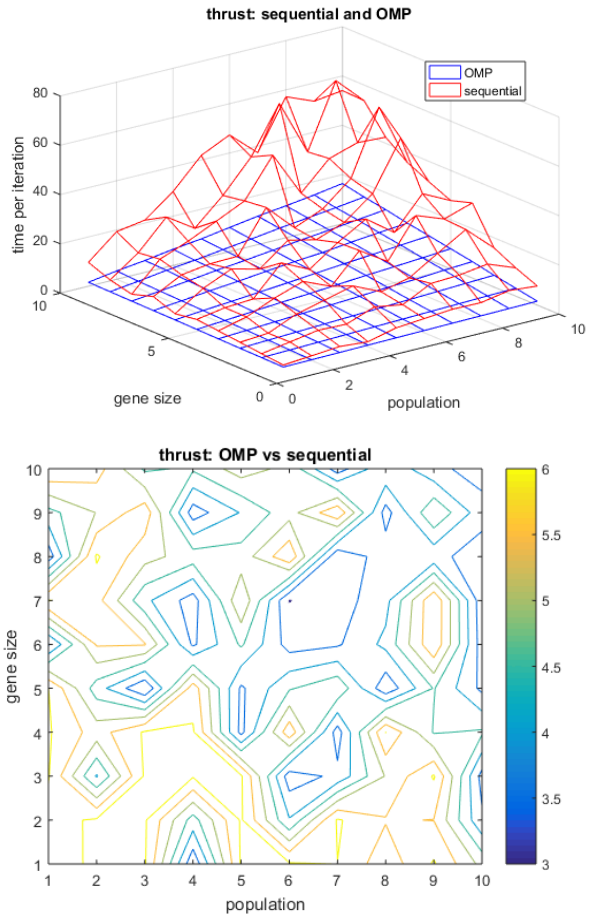

Figure 3: Ackley Function OMP version

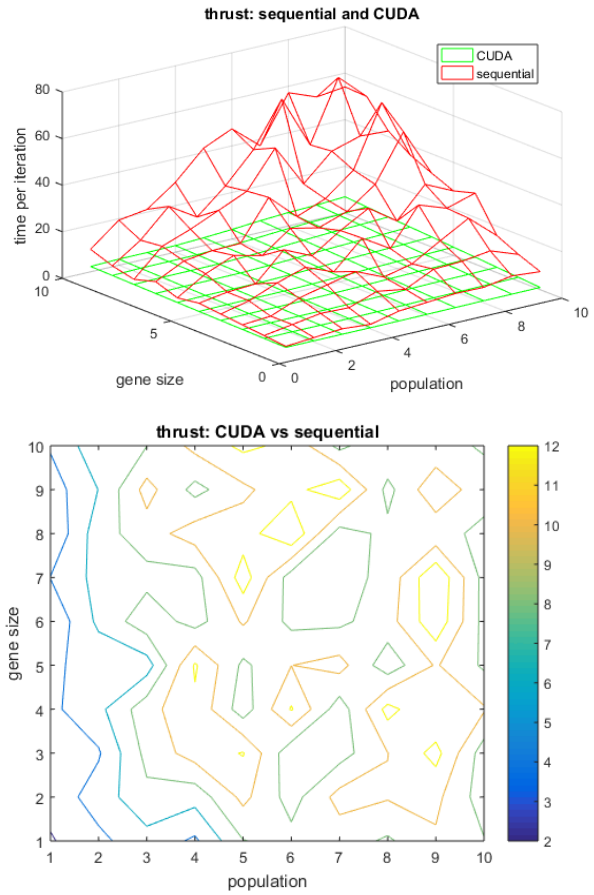

Figure 4: Ackley Function CUDA version

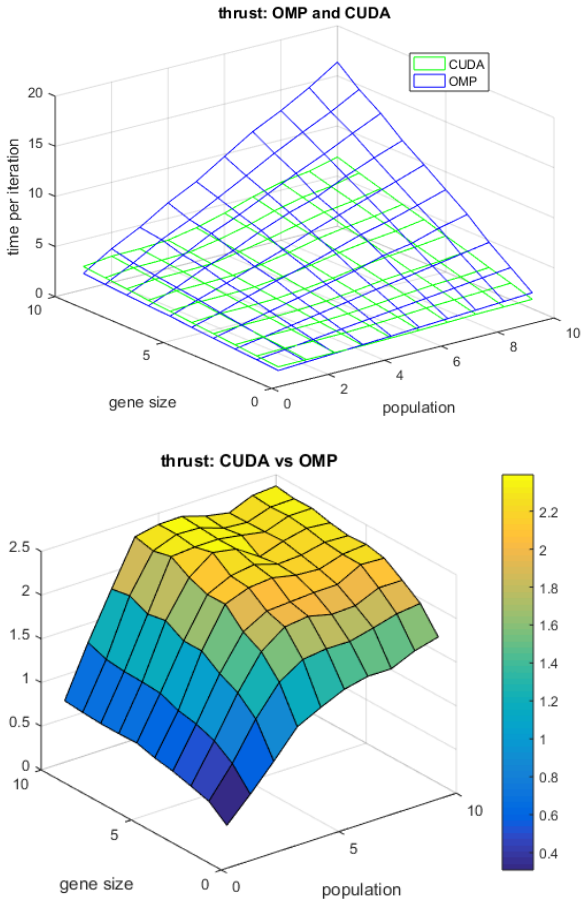

Figure 5: Ackley Function OMP vs. CUDA

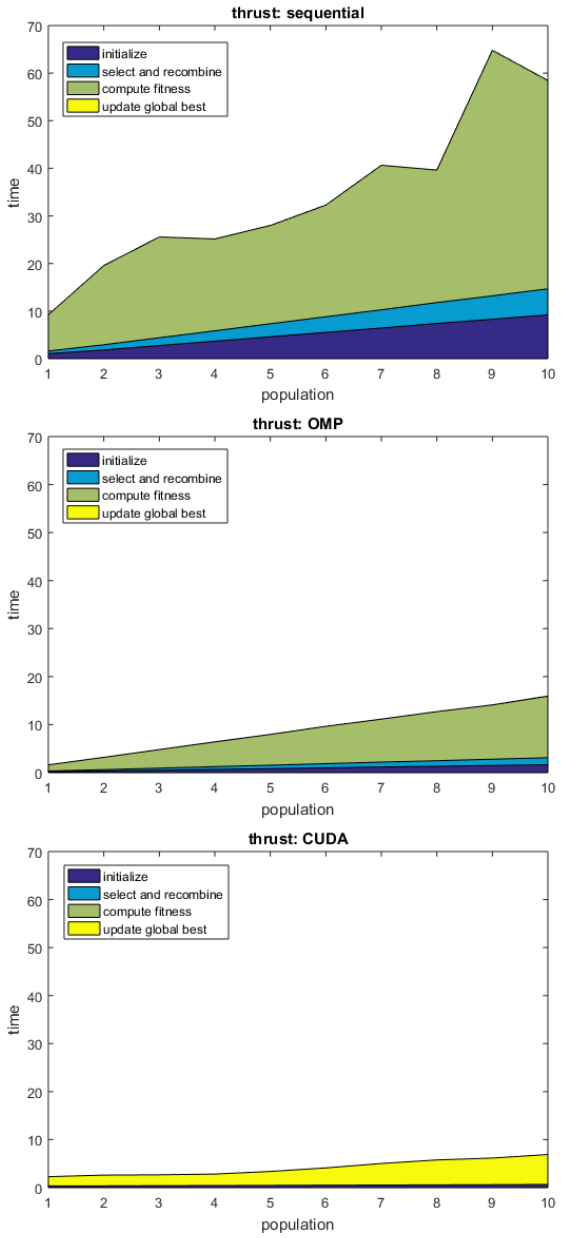

Figure 6: Ackley 時間分割圖

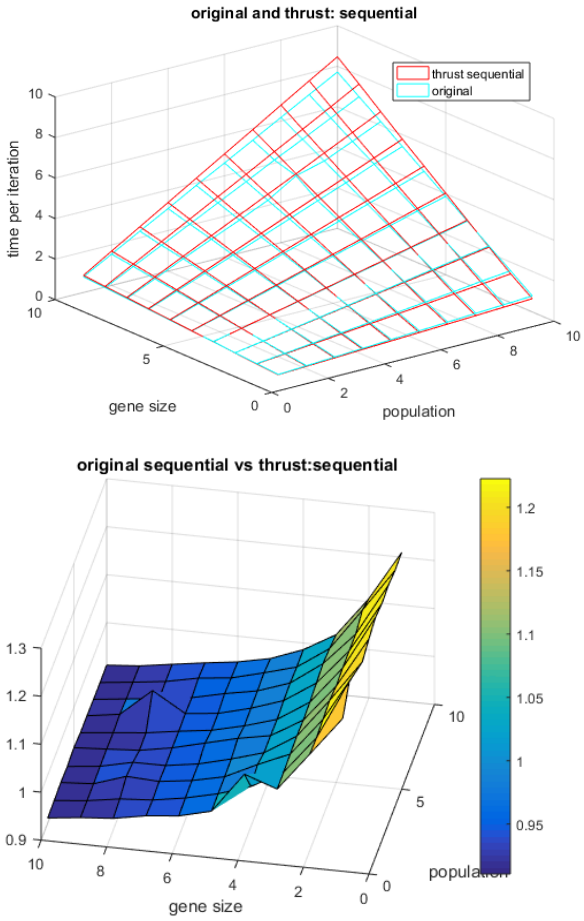

Figure 7: OneMax Function serial version

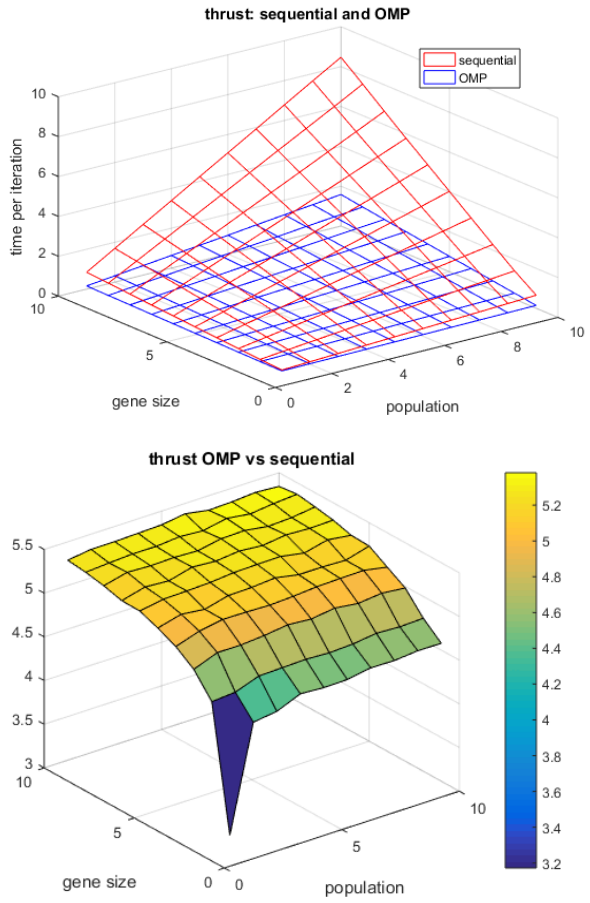

Figure 8: OneMax Function OMP version

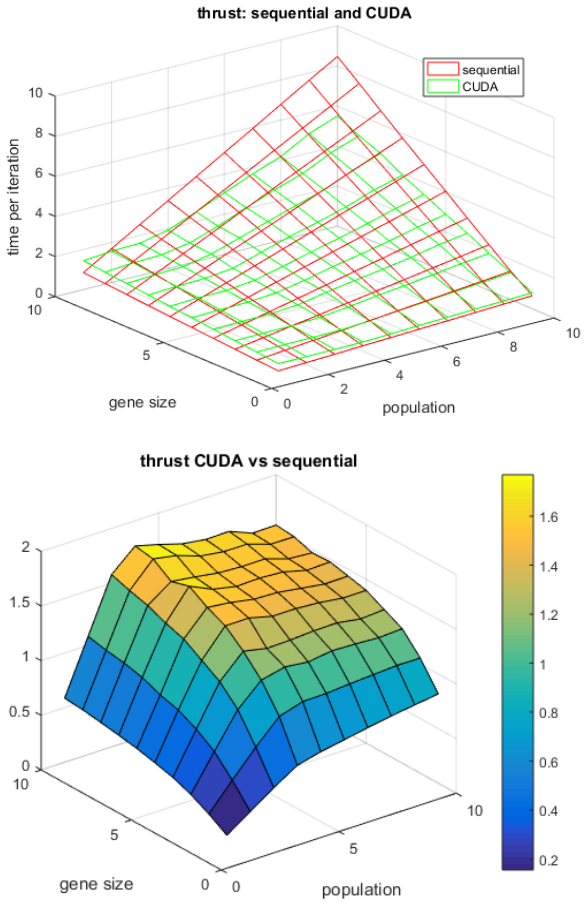

Figure 9: OneMax Function CUDA version

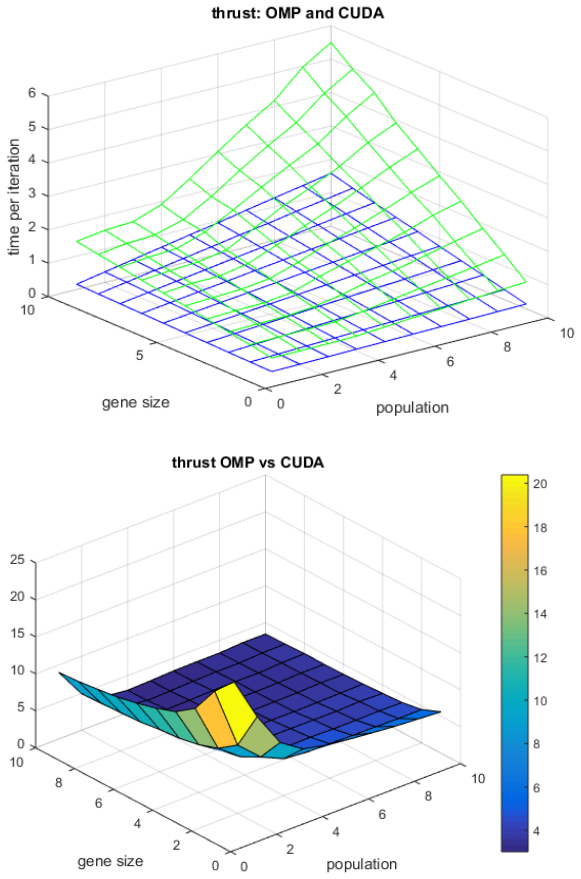

Figure 10: OneMax Function OMP vs. CUDA

（上）time per iteration
（下）OMP/CUDA 的時間比值。

4.2.5 時間分割圖

在這個問題，sequential 表現得很平穩，而 OMP 也有顯著的效果。但 CUDA 花費了非常大量的時間在 update global best 上。

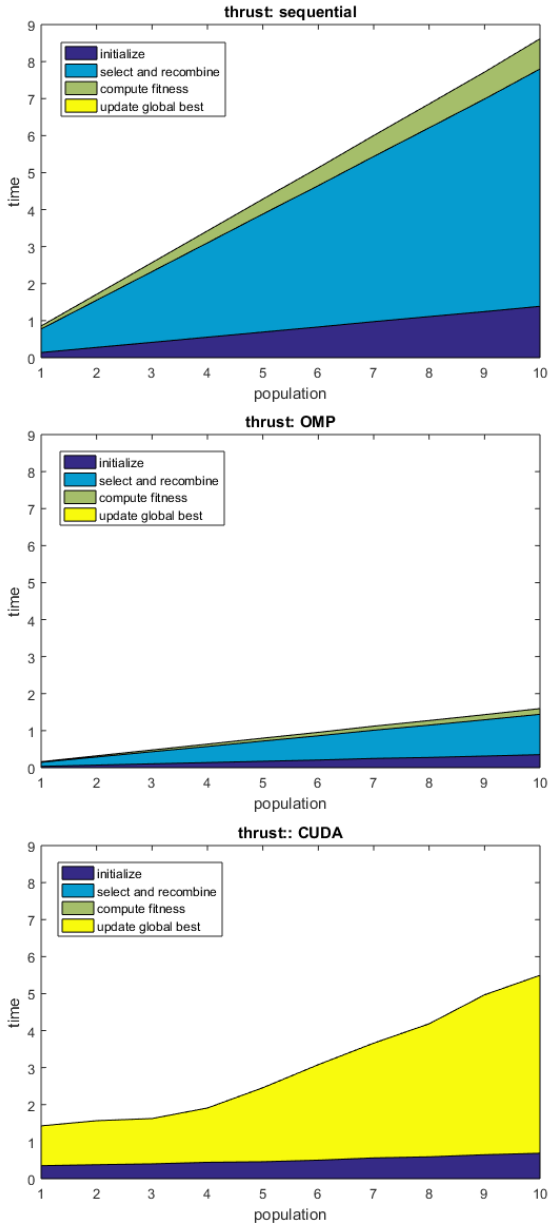

Figure 11: OneMax Function 時間分割圖

5. RELATED WORK

在過去，已經有許多平行化基因演算法的模型被提出，大致可分為：(1) Global single-population master-slave GAs，(2) single-population fine-grained GAs，(3) multiple-population coarse-grained GAs [1]。有人嘗試以 CUDA 實作上述第三種方法，證實平行化可以大幅提升基因演算法的效率 [2]。

6. REFERENCES

[1] Cantú-Paz, E. A survey of parallel genetic algorithms. *CALCULATEURS PARALLELES, RESEAUX ET SYSTEMS REPARTIS 10* (1998).

[2] Pospichal, P., Jaros, J., and Schwarz, J. *Parallel Genetic Algorithm on the CUDA Architecture*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 442–451.