基於單側雅可比演算法於 CUDA 計算機實現平行矩陣特徵值分解

宋啟嘉* 胡逸秀
國立虎尾科技大學電機工程學系
*ccsun@nfu.edu.tw

摘　　要

本文提出基於單側雅可比演算法實現於支援平行編譯與計算平台(CUDA)達到硬體加速優化。在現今的計算系統中，由於單一個 CPU 的核心效能已經漸漸達到硬體上的限制，而 GPU 計算加速架構的優勢在於它本身固有的平行特徵，故在多核心平行計算的環境之下更能突顯其優勢。另一方面，特徵值分解(EVD)在通訊系統中長期扮演著一個非常重要的角色，因為在通訊系統中常以矩陣特徵值作為訊號通道判斷的依據，故實現平行特徵值分解計算有其重要性。經由實現平行單側雅可比演算法於 CUDA 計算機後，由於單側雅可比演算法本身簡單、快速收斂以及固有的平行特徵，使輸入之對稱矩陣能夠經過連續疊代計算分解為特徵值與特徵向量，最後實驗結果顯示本文所提出的單側雅可比演算法於 CUDA 計算機上之效能與傳統循環雅可比演算法比較，其計算速度快近 102~103 倍之多。

關鍵詞： 平行矩陣特徵值分解，GPU 計算，單側雅可比演算法，CUDA

One-Sided Jacobi Algorithm based on CUDA Computer for Parallel Eigenvalue Decomposition

Chi-Chia Sun* Yi-Siou Hu
Department of Electrical Engineering, National Formosa University
**ccsun@nfu.edu.tw*

ABSTRACT

In this paper, One-Sided Jacobi algorithm is optimized on the Graphics Processing Unit (GPU) for hardware acceleration using Compute Unified Device Architecture (CUDA) computer. In modern computation systems, due to the fact that the performance of a single CPU core has already reached its limitation, the architecture of GPU with many cores is advantageous in its inherent parallel features and extremely fast computation time. In order to realize high performance parallel Eigenvalue Decomposition (EVD) on the CUDA computer, One-Sided Jacobi parallel algorithm is selected, due to its simple, strong convergence and inherent parallel features. It can factorize the input symmetric matrix into eigenvalues and eigenvectors iteratively. Finally, the experimental results show that the performance of the proposed One-Sided Jacobi Method using CUDA hardware accelerator is improved to 102~103 times faster than the conventional Cycle-by-Row Jacobi algorithm.

Keywords: Parallel EVD，GPU Computing，One-side Jacobi algorithm，CUDA

文稿收件日期 104.4.13;文稿修正後接受日期 105.8.30; *通訊作者
Manuscript received April 13, 2016; revised August 30, 2016; * Corresponding author

一、前　言

近年來，由於現代圖形處理器(GPU)強大的並行處理能力和可程式流水線，使用 GPU 與搭配 CPU 漸漸廣泛被用來實現科學計算加速、工程分析模擬和大數據分析之上[1-6]。特別在面對單指令流多數據流(SIMD)的計算，且數據處理的運算量遠大於數據調度和傳輸的需要情況之下，GPU 圖形處理器在性能上會超越了大部份傳統的中央處理器應用程式。目前，多核心中央處理器計算效能約略在每秒 60~100 GFlops 上下，而 GPU 藉由將內處理器串通起來，使內部的各個內處理器能夠交換、同步和共享資料、同步和共享資料，成為多執行緒處理器去解決資料密集的計算，單一 GPU 計算單元的效能已可以達到每秒 2~5 TFlops 的計算效能。此外，GPU 計算亦已被用於許多平行矩陣計算應用中，例如共軛梯度計算[7]，QR 矩陣分解平行計算 [8]，Wavelet 小波轉換等等[9]。

在數值線性代數中，雅可比法(Jacobi Method)是一種解對角元素幾乎都是各行和各列絕對值為最大值的線性方程組演算法，求解出每個對角元素並插入近似值。其中不斷迭代直至收斂的雅可比疊代法(Jacobi Iterative Method)已非常廣泛被應用在平行處理來完成快速分解對稱矩陣特徵值(EVD)，與另外一種被普遍使用的 QR 分解法(QRD)比較，雅可比疊代法的最大優勢則是即便在運算過程中因為硬體簡化或是遭遇到計算錯誤而造成結果有誤差值存在的情況發生，雅可比疊代法依然能夠計算出可接受範圍內的矩陣特徵值[10-12, 20-21]。

在本論文中，為了實現雅可比疊代法的 GPU 平行計算設計，進一步提高其計算效能，循環雅可比方法與單側雅可比方法分別被實現於 NVIDA GTX-7700 圖形處理器的 CUDA 計算機平台之上，從實驗數據分析中發現，雖然單側雅可比方法在演算法上需要一直保持同一個矩陣計算維度，但也正是此演算法因為不需要交換計算維度的特色，實際上大大的降低CUDA 計算機中 CPU Host 端與 GPU Device 端之間的交換資料量。單側雅可比演算法於 CUDA 計算機上之效能與傳統循環雅可比演算法比較，其計算速度快近 10^2~10^3 倍之多。若與一般的處理器比較，可以達到約 10^2 倍的計算速度。

本論文的結構如下，第 2 節首先介紹雅可比矩陣特徵值分解方法，第 3 節將會對如何於 CUDA 計算機實現平行單側雅可比方法進行討論。第 4 章節為實驗數據結果，而第 5 章節為本文的結論。

二、雅可比矩陣特徵值分解方法

2.1 對稱矩陣分解

特徵值分解為矩陣分解應用中最廣泛常見的方法[13]，假設 A 為對稱矩陣，而以下公式也成立，λ 則是特徵值(包含一個或以上的非零元素)，V 被定義為特徵向量

$$AV = \lambda V \tag{1}$$

經由代換之後式子等效如下：

$$A - \lambda I = 0 \tag{2}$$

I 為單位矩陣，此公式亦被稱作為矩陣 A 的特徵多項式，根據其定義特徵向量無法為零矩陣，為了使這個解存在，$A - \lambda I$ 必須為奇異的，換句話說，它的行列式必須為零，其目的在於找出公式 3 的特徵值。

$$|A - \lambda V| = 0 \tag{3}$$

特徵值 $D = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ 與矩陣 A 相對應到線性獨立的特徵矩陣 V 可以被描述為:

$$A = VDV^{-1} \tag{4}$$

2.2 循環雅可比演算法

一個大小為 $n \times n$ 的對稱矩陣 A 的特徵值分解可分解成三個矩陣來表示 $A = VDV^T$，V 是正交矩陣 $(VV^T = I)$，D 則是對角矩陣，其中 D 為矩陣 A 的特徵值，雅可比演算法是將對稱矩陣經由雅可比旋轉改變其角度並以迭代的方式實現近似特徵值分解[14][15]，進而求得特徵值及特徵向量，可表示如下：

$$A_{k+1} = V_k A_k V_k^T, k = 0, 1, 2, \cdots, \tag{5}$$

循環雅可比演算法之整體程式架構可參考圖 1 的虛擬程式碼，而在這段程式碼中第 22 行為矩陣乘法，第 19 行為呼叫雅可比旋轉動作，其雅可比旋轉之函數虛擬程式碼如圖 2 所示為，在圖 2 中第 11 行至 14 行雅可比旋轉的數學式。

```
1   PROCEDURE   Cycle_by_row   IS
2     Input: Symmetric Matrix = A
3     Input: Error Minimum = tol
4     Output: Diagonal Matrix = A
5     Output: Sweeps = sweep
6   BEGIN
7
8       [n, n]=size(A);
9       ew=sort(eig(A));
10      Id=eye(n, n);
11      V=Id;
12      nA=norm(A, 'fro');
13      ep=tol*nA;
14      sweep=1;
15
16      WHILE offA > ep
17        FOR i IN 0 TO n-1 LOOP
18            FOR j IN i+1 TO n LOOP
19                [Ja, tau] = Jacrot(A, i, j);
20                G=Id;
21                G([p q],[p q])=J;
22                A=G'*A*G;
23                V=V*G;
24          END LOOP
25        END
26        offA=norm(A-diag(A), 'fro');
27        sweep=sweep+1;
28      END WHILE
29  END PROCEDURE;
```

圖 1 循環雅可比演算法的虛擬程式碼

```
1   PROCEDURE    Jacrot    IS
2     Input: Symmetric Matrix = A
3     Input: Row = i
4     Input: Column = j
5     Output: Jacobi Rotation Matrix = Ja
6   BEGIN
7     IF A(i,j)==0 THEN
8       c=1;
9       s=0;
10    ELSE
11      tau= (A(j, j)-A(i, i))/(2*A(i, j));
12      t= sign(tau)/(abs(tau)+sqrt(1+tau^2));
13      c= 1/sqrt(1+t^2);
14      s=t*c;
15    END IF
16    Ja=[c s; -s c];
17  END PROCEDURE;
```

圖 2 雅可比旋轉的虛擬程式碼

而V_k是個正交旋轉矩陣，其旋轉角度是在(i, j)座標的角度θ，其為特性矩陣並包含四個非零元素$V_{ii}=V_{ij}=\cos\theta_k$和$V_{ij}=-V_{ji}=\sin\theta_k$：

$$\begin{pmatrix} 1 & 0 & & & & 0 & \\ 0 & & & \cdots & & & \\ & & \cos\theta_k & & \sin\theta_k & & \leftarrow row\ i \\ \vdots & & & \ddots & & \vdots & \\ & & -\sin\theta_k & & \cos\theta_k & & \leftarrow row\ j \\ & & & \cdots & & & \\ 0 & & \uparrow & & \uparrow & 1 & \\ & & col\ i & & col\ j & & \end{pmatrix} \quad (6)$$

雅可比正交旋轉矩陣$V_k(k=0,1,2,3,\ldots,)$可由不同的執行順序來疊代計算出矩陣特徵值，最常見的方法為循環雅可比演算法[16-17]，而座標(i,j)選取的順序為：

$$(i,j)=(1,2)(1,3)\ldots(1,n)(2,3)\ldots(2,n)\ldots(n-1,n) \quad (7)$$

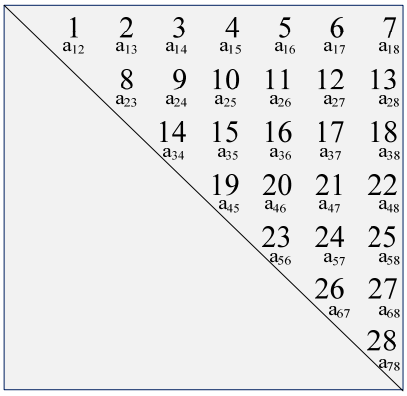

圖 3 循環雅可比演算法之選取順序(矩陣階數8×8為例)

當$N=n(n-1)/2$個所對應到的相對(i, j)索引值執行完後稱作一次sweep掃描，如圖3所示，a_{ij}為矩陣元素位置，數字則為執行順序，經過數次sweep掃描計算，矩陣A將會逐漸收斂成一個對角矩陣D，對角矩陣D將會包含矩陣A特徵值$\lambda_1,\lambda_2,\ldots,\lambda_n$：

$$\lim_{k\to\infty}A_k=diag[\lambda_1,\lambda_2,\ldots,\lambda_n]=\begin{pmatrix} \lambda_1 & 0 & & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & & 0 \\ 0 & 0 & \cdots & 0 & \lambda_n \end{pmatrix} \quad (8)$$

循環雅可比演算法對處理對稱矩陣特徵

```
1   PROCEDURE   One_Sided   IS
2         Input: Symmetric Matrix = A
3         Input: Error Minimum = tol
4         Output: Diagonal Matrix = A
5         Output: Sweeps = sweep
6   BEGIN
7         [Uori, Dori]=eig(A);
8         [n,n]=size(A);
9         Id=eye(n,n);
10        V=Id;
11        nA=norm(A,'fro');
12        ep=tol*nA;
13        sweep=1;
14        tmp=zeros(n,2);
15
16  WHILE offA > ep
17     FOR i IN 1 TO n LOOP
18        FOR j IN i+1 TO n LOOP
19           p_norm=norm(A(:,i),2);
20           q_norm=norm(A(:,j),2);
21           r=A(:,i)'*A(:,j);
22           tmp=[A(:,i),A(:,j)];
23           tau=(q_norm^2 - p_norm^2)/(2*r);
24           t=(sign(tau))/(abs(tau)+sqrt(1+tau^2));
25           c=1/sqrt(1+t^2);
26           s=t*c;
27           Ja=[c s; -s c];
28           tmp= [A(:,i),A(:,j)]*Ja;
29
30           IF p_norm > q_norm THEN
31              A(:,i)=tmp(:,1);
32              A(:,j)=tmp(:,2);
33           ELSE THEN
34              A(:,i)=tmp(:,2);
35              A(:,j)=tmp(:,1);
36           END IF
37        END FOR
38     END FOR
39     offA=norm(A-diag(A), ‘fro’);
40     sweep=sweep+1;
41  END WHILE
42  END PROCEDURE;
```

圖 4 單側雅可比演算法之虛擬程式碼

值問題可以很容易實現於 CUDA 之上，其原因在於演算法本身所具備的平行計算特色。另一方面，由於雅可比演算法式主要是透過實現連續 $A \leftarrow V^T AV$ 的正交矩陣乘法來逼近特徵值，在計算過程中非對角矩陣元素數值會漸漸收斂至對角矩陣元素之上。由於雅可比演算法是經由矩陣迭代特徵值分解，故無法事先得知迭代運算會在何時結束，但由於雅可比運算會逐漸將非矩陣斜角元素往矩陣的對角線集中，最後會產生斜角矩陣 D，而矩陣 D 上的斜角矩陣元素即為 A 矩陣的特徵值，同時，非矩陣斜角元素會逐漸逼近於零。換句話說，特徵值的收斂程度可以透過觀察非矩陣斜角元素之範數值:

$$off(A) = \sqrt{\sum_{i=1}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}^2} \tag{9}$$

當此範數值越小代表著該矩陣的非矩陣斜角之元素平方總和的值也越趨近於零，直到非矩陣斜角元素範數低於預先設定的閥值 tol，此矩陣迭代特徵值分解就會停止，如虛擬程式碼第 16 行所示。

2.3 單側雅可比演算法

循環雅可比演算法已可在 CUDA 計算機實現硬體平行加速[18-19]，但由於循環雅可比演算法的每次疊代計算部分都要先做一次二維矩陣乘法運算，接著做一次轉置矩陣計算，然後接著再做一次二維矩陣乘法運算，矩陣轉置運算此方法實現於 GPU 平行計算時必須透過 CPU Host 端來回控制 GPU Device 與 CPU Host 兩端的資料，最後再將值傳回 CPU 端，每次疊代計算逼近矩陣特徵值時不斷地重複此動作，循環雅可比演算法會使得大部分計算時間浪費在 GPU 與 CPU 兩端的資料交換，且計算時間會隨著矩陣增大而相對地增加[18]。因此，本論文提出較計算方法較為簡單的單側雅可比演算法，雖然單側雅可比演算法整體計算複雜度較循環雅可比演算法稍高，但因為此方法只使用單次的一維矩陣乘法運算，且計算過程中不需要執行矩陣轉置運算，換句話說，單側雅可比演算法實現於 GPU 平行計算時可大幅減少 CPU 端來回控制 GPU 與 CPU 兩端的資料量，還能在更短的時間完成特徵值分解。

圖 4 為單側雅可比演算法之整體虛擬程式碼架構，第 19 至 21 行為單側雅可比演算法之計算 P_{norm}、Q_{norm} 及 r 值用於單側雅可比旋轉所需之參數計算，而第 23 至 27 行為單側雅可比旋轉，與圖 3 中循環雅可比旋轉計算方法不同，第 28 行則為單側雅可比旋轉的矩陣乘法部分，第 30 至 36 行為 swap 功能，主要是將矩陣中該行之範數最大的放置矩陣前面的位置。與循環雅可以演算法比較起來，其計算複雜度較循環雅可比演算法稍高，而矩陣分解之精度則相同。

三、單側雅可比演算法之平行運算

實現平行運算最主要目的就是當有很大的運算量需求要求在最短時間內盡可能完成運算的話，勢必要將運算資料適當的分配，以達到各個分算運算可以同步進行，換句話說就是在同時間內增加執行緒數量用以加快運算速度。當矩陣大小為 n，則單側雅可比演算法每次的矩陣乘法運算為 $n\times 2$ 與 2×2 的方式執行，故在此本文使用了 $n\times 2$ 個平行執行緒 Parallel Threads 分配其各自的工作，其中平行工作內容有計算 P_{norm}、Q_{norm} 及 r 值、雅可比旋轉角度及矩陣乘法運算，每一項分散計算工作都必須適當的安排以求達到執行緒數量最大化，CUDA 計算機執行時每一個執行緒 Thread 都是獨立且同步的進行，詳細的執行緒 Thread 工作分配之說明可參考表 1。如圖 5 所示，假設以矩陣大小為 $n=8$ 的對稱矩陣為例，在此將以上所提的各種數值運算分配給予 Threads，dim3 threads(tx, ty)為 Grid 宣告，而 tx 與 ty 皆為 Threads，以此例說明 threads(2, 8)，平行執行緒數量共有 $8\times 2=16$ 個，每個元素分配一個執行緒 Thread 使其可以執行同步運算，同時也能給予每個執行緒 Thread 執行不同的運算。

表 1 執行緒 Thread 平行運算工作分配

tx==0	ty==0~7	分別計算P_{norm}與r的乘法運算 P_{norm}：將i行的各個元素平方，並存至暫存器 r：將i、j行之同列元素相乘，並存至暫存器
tx==1	ty==0~7	Q_{norm}：將j行的各個元素平方，並存至暫存器
呼叫CUDA指令函數_syncthreads()		
tx==0	ty==0	將i行平方後之值做累加運算後開平方根號
tx==0	ty==1	將j行平方後之值做累加運算後開平方根號
tx==0	ty==2	將r暫存器之所有值做累加運算
呼叫CUDA指令函數_syncthreads()		
tx==0	ty==0	雅可比旋轉之角度運算
呼叫CUDA指令函數_syncthreads()		
tx==0	ty==0~7	計算矩陣乘法中 i 行的 a'_{11}, a'_{21},…, a'_{81}部分
tx==1	ty==0~7	計算矩陣乘法中 j 行的 a'_{12}, a'_{22},…, a'_{82}部分
呼叫CUDA指令函數_syncthreads()		

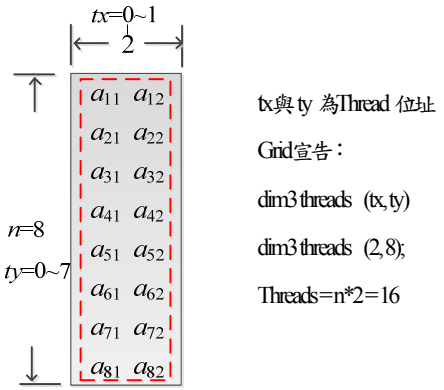

圖 5 執行緒 Thread 於矩陣分解平行運算的分配關係

為了詳細說明 Threads 的運算工作分配，在此以圖 6 表示在 8 個執行緒 Threads 為 $tx==0\,\&\,ty==0\sim 7$ 時計算 P_{norm} 與 r 值的乘法運算，而在另外 8 個執行緒 Threads 為 $tx==1\,\&\,ty==0\sim 7$ 時計算 Q_{norm} 的乘法運算，此運算後面必須使用 CUDA 指令函數_syncthreads()，在此個別執行緒 Thread 分別完成計算後會互相等待直到 16 個平行執行緒 Threads 皆完成計算為止。

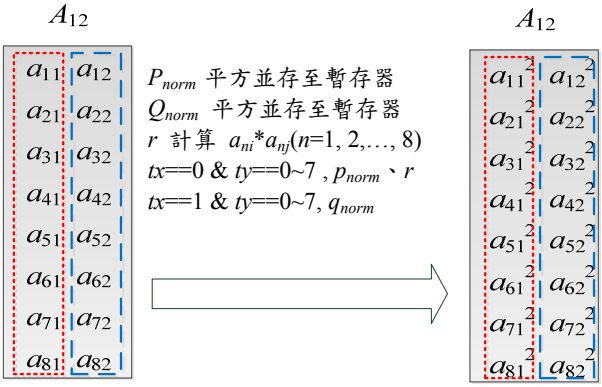

圖 6 平行運算 Pnorm、Qnorm 及 r 之 Threads 分配

接著，圖 7 表示在 Threads 為 $tx==1\,\&\,ty==0$ 、$tx==0\,\&\,ty==1$ 時計算 P_{norm} 、Q_{norm} 累加及開根號運算，而 $tx==0\,\&\,ty==2$ 時計算 r 值的累加運算，此運算後使用 CUDA 指令函數_syncthreads()，等待直到 3 個平行執行緒 Threads 皆完成計算為止，在這裡為止對應到單側雅可比演算法虛擬程式碼的第 19 至 22 行。

。

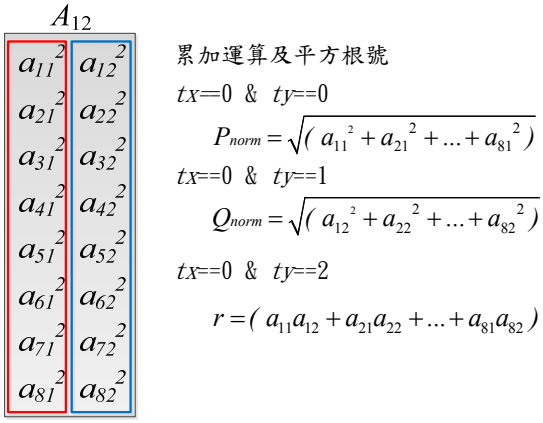

圖 7 平行加法及根號運算之 Threads 分配

接下來 $tx==0$ & $ty==0$ 時，使用單一執行緒 Thread 計算雅可比旋轉角度，接著呼叫 CUDA 指令函數_syncthreads()，等待此單一平行執行緒 Threads 完成計算，在這裡則對應到單側雅可比演算法虛擬程式碼的第 23 至 26 行。最後，圖 8 則說明 $n\times 2$ 個平行執行緒 Threads 執行矩陣乘法運算，當 tx==0 & ty==0~7 執行矩陣乘法中的 $a'_{11},a'_{21},\ldots,a'_{81}$ 的部分。當 tx==1 & ty==0~7 則實質行矩陣乘法中的 $a'_{12},a'_{22},\ldots,a'_{82}$ 的部分，並呼叫 CUDA 指令函數_syncthreads()等待直到 $n\times 2$=16 個平行執行緒 Threads 皆完成計算為止，在這裡則對應到單側雅可比演算法虛擬程式碼的第 27 至 36 行。

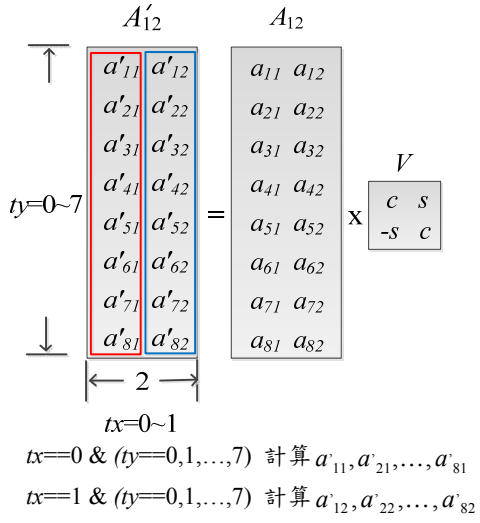

圖 8 平行矩陣乘法運算之 Threads 分配

四、實驗結果

本論文的模擬環境包含 PC 模擬環境與 CUDA 模擬環境，PC 環境下使用 MATLAB 進行模擬分析，而 CUDA 計算機使用 Visual Studio 2008 C++實現的平行運算性能分析，演算法計算時間僅包含雅可比演算法的疊代迴圈部分，不包含產生隨機矩陣、開讀讀檔等其他非雅可比演算法的運算部分。

模擬環境:

- 軟體　：CUDA 5.5、Matlab 2012B
- 顯示卡：NVIDIA GeForce GTX770
- 處理器：Intel i3　3.2 GHz
- 記憶體：DDR III 8G

本文所提出適用於 CUDA 平行計算機架構的單側雅可比演算法與循環雅可比演算法分別用矩陣階數為 10、20、40、80 及 160 的隨機產生對稱矩陣來比較計算速度，此兩種演算法分別實現於 Matlab 與 CUDA 計算機，亦即是分別比較兩種演算法在不同的矩陣階數於 CPU 與 GPU 下的計算速度。特徵值分解之計算使用單側雅可比演算法實現於 Matlab 程式碼，另一方面，同樣的架構也被實現於 CUDA 計算機之上，由於 CUDA 前置處理只可能實現於 CPU Host 端，GPU Device 端受控於 Host 端，所有 Device 端的計算值最後還要被傳回 Host 端，新的平行工作將由 Host 端指派給 Device 端，在此 CUDA 平行程序中有個缺點就是在 Host 端與 Device 端中資料傳送速度很慢，基於這個原因，實現循環雅可比演算法於 CUDA 計算機架構將會是較不理想的[19]。圖 9 所示為循環雅可比演算法於 CPU 及 CUDA 比較不同的矩陣階數，分別為 10、20、40、80 及 160 時的計算速度。由於循環雅可比演算法於 GPU 之計算時間龐大，故在此使用對數型態表示。由圖 9 中可發現循環雅可比演算法於 GPU 之計算時間遠大於在 CPU 計算，其原因在於此循環雅可比演算法本身有些計算不適合 CUDA 計算機如：矩陣轉置，此運算將會嚴重影響 CUDA 之計算速度，每做一次雅可比旋轉角度計算就需要執行矩陣轉置，此過程需要不斷地在 CPU 與 GPU 中進行資料傳輸，故消耗許多時間，因此循環雅可比演算法較不適用於 GPU 之 CUDA 計算。圖 10 所示為單側雅可比演算法於 CPU 及 CUDA 比較不同的矩陣階數，分別為 10、20、40、80

及 160 時的計算速度。很明顯可以發現到單側雅可比演算法於 GPU 時之計算速度可提升，在矩陣階數 80 可以達到將近 10^2 倍。

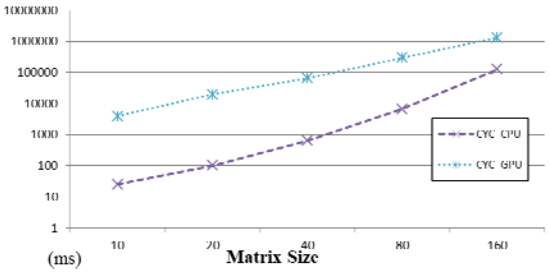

圖 9 比較循環雅可比演算法於 CPU 及 GPU 的計算速度。

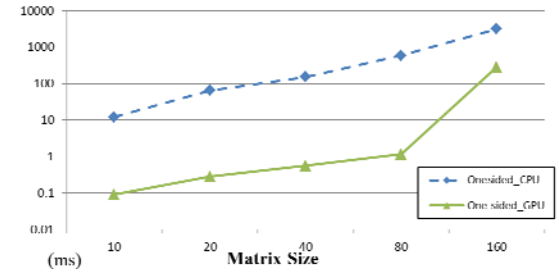

圖 10 比較單側及循環雅可比演算法於 CPU 及 GPU 的計算速度。

圖 11 為單側雅可比演算法及循環雅可比演算法於 CPU 與 CUDA 時之計算速度，由圖 11 可得知單側雅可比演算法之計算時間遠小於循環雅可比演算法，無論是在矩陣階數 10 至 160 皆有明顯的差距。比較單側雅可比演算法與循環雅可比演算法分別於 GPU 與 CPU 時的計算速度，可以發現單側雅可比演算法的計算時間明顯小於循環雅可比演算法，尤其是當單側雅可比演算法於 GPU 且矩陣階數為 160 時其計算速度可比循環雅可比演算法快將近 10^3 倍，也比循環雅可比演算法之 CPU 快將近 10^2 倍，因此本文所提出的基於單側雅可比演算法於 CUDA 計算機實現平行矩陣特徵值分解明顯較循環雅可比演算法來的快速。

同時我們也發現單側雅可比演算法 CUDA 計算機的計算速度仍然受限於 GPU 之 CUDA 核心數目，目前使用的 GTX770 的核心數目可以達到最高效率之平行計算為 1536 個 Threads，換句話說使用此 CUDA 平行處理器核心可以即時處理最大 768×768 之矩陣，若矩陣階數大於 CUDA 平行處理器核心數，則將以虛擬方式代替不足的核心數，其平行效果則不佳甚至計算速度較一般 CPU 慢。

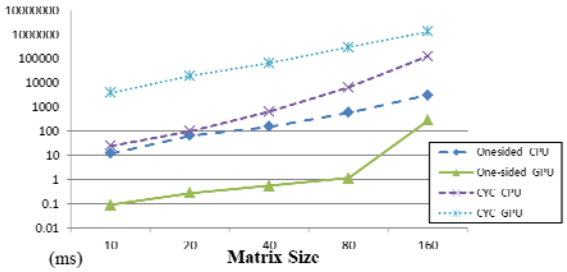

圖11 比較單側及循環雅可比演算法於GPU與CPU的計算速度。

五、結論

本文提出基於單側雅可比演算法與 CUDA 計算機實現矩陣特徵值分解平行計算，實驗數據顯示單側雅可比演算法較循環雅可比演算法來的更合適實現於 GPU 硬體加速平台，主要原因為單側雅可比演算法的一維矩陣乘法疊代運算方法省去了循環雅可比演算法的二維矩陣乘法需要較複雜的轉置運算，而簡單的一維矩陣乘法運算使得程序員可以將大部分的平行執行序直接實現於 CUDA 計算機上，CPU Host 端只需要負責輸入一開始的隨機對稱矩陣與等待矩陣特徵值分解的結果，換句話說，單側雅可比演算法可以有效減少了 CPU Host 端與 CUDA Device 端之間傳輸資料的次數，同時也大量減少了不必要的運算時間，其計算速度比循環雅可比演算法快將近 10^2~10^3 倍之多。

參考文獻

[1] NVIDIA, “CUDA C Programming Guide, http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz37gu1mIrl”, July 2014

[2] D. B. Kirk and W. W. Hwu, “Programming Massively Parallel Processors”, Morgan Kaufmann, Feb. 2010.

[3] NVIDIA, “NVIDIA CUDA Compute Unified Device Architecture Reference Manual”, June 2008.

[4] 張舒、褚艷利、趙開勇與張鈺勃，“GPU 高效能運算之CUD”，碁峯資訊股份有限

公司，2011年七月出版。
[5] S. Tariq, “An Introduction to GPU Computing and CUDA Architecture”, NVIDIA corporation, July 2011.
[6] J. Ghorpade, J. Parande, M. Kulkarni and A. Bawaskar, “GPGPU Processing in CUDA Architecture,” Advanced Computing: An International Journal, vol.3, no.1, Jan. 2012.
[7] M. Maleko, “A Jacobi Algorithm for Simultaneous Diagonalization of Several Symmetric Matrices,” Technical report, Royal Institute of Technology, Jan. 2003.
[8] J. Sanders and E. Kandrot, “CUDA by Example”, Addison-Wesley, July 2010.
[9] P. B. Richard and T. L. Franklin, “The solution of singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays,” SIAM journal on Scientific and Statistical Computing, vol. 6, no. 1, pp. 69-84, Jan. 1985.
[10] W. Yang, “Linear Algebra, Surveying and Mapping Press”, April 2008.
[11] D. H. PHAM and T. WADA, “A Novel Fast Eigenvalue Decomposition based on Cyclic Jacobi rotation and its application in eigen-beamforming,” Technical Report of IEICE, vol. 106, no. 188, pp. 903-0129, July 2006.
[12] J. Goetze, S. Paul, and M. Sauer, “An Efficient Jacobi-Like Algorithm for Parallel Eigenvalue Computation,” IEEE Transactions on Computers, vol. 42, no. 9, pp. 1058-1065, Sep.1993.
[13] G. W. Stewart, “Matrix algorithms”, Dec. 2001.
[14] G. H. Golub, V. Loan and F. Charles, “Matrix computations”, Johns Hopkins University Press, Oct. 1996.
[15] M. Maleko, “A Jacobi Algorithm for Simultaneous Diagonalization of Several Symmetric Matrices,” Technical report, Royal Institute of Technology, Jan. 2003.
[16] K. Veselic, “A Jacobi eigenreduction algorithm for definite matrix pairs,” Numerische Mathematik, vol. 64, no. 1, pp. 241-269, July 1993.
[17] Y. Liu, C. S. Bouganis and P.Y.K. Cheung, “Hardware architectures for eigenvalue computation of real symmetric matrices,” The institution of Engineering and Technology, vol. 3, no. 1, pp. 72-84, Jan. 2009.
[18] C. C. Sun and Y. S. Hu, “One-Dimension Parallel Jacobi Method on the GPU using CUDA,” ICIC Express Letters, vol. 7, no. 10, pp. 2753-2758, Oct. 2013.
[19] T. Wang, L. Guo, G. Li. and J. Li, “Implementing the Jacobi Algorithm for Solving Eigenvalues of Symmetric Matrices with CUDA,” IEEE Seventh International Conference on Networking, Architecture, and Storage, June 2012, pp. 69-78.
[20] Zlatko Drmač and Krešimir Veselić, “New Fast and Accurate Jacobi SVD Algorithm. I,” SIAM Journal on Matrix Analysis and Applications, vol. 29 no.4 pp. 1322-1342. 2008.
[21] Martin Becka and Marian Vajtersic, “Block-Jacobi SVD algorithms for Distributed Memory System I: Hypercubes and Rings”, Parallel Algorithms and Applications, vol. 13 no.3 pp. 265-287 1999.