國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

運用圖形處理器增進計算巴黎選擇權價格的效能

Using GPU To Accelerate the Pricing of Parisian Options

張智烜
Chang Chih-Hsuan

指導教授：呂育道 博士
Advisor: Lyuu Yuh-Dauh, Ph.D.

中華民國 99 年 7 月

July, 2010

國立臺灣大學碩士學位論文
口試委員會審定書

運用圖形處理器增進計算巴黎選擇權價格的效能

Using GPU To Accelerate the Pricing of Parisian Options

本論文係張智烜君（R96922087）在國立臺灣大學資訊工程學所完成之碩士學位論文，於民國 99 年 7 月 5 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

_____（簽名）

（指導教授）

_____ _____

_____ _____

_____ _____

_____ _____

系主任、所長 _____（簽名）

誌謝

感謝呂育道教授在這期間內耐心地給我指導和提供意見，讓我能順利完成研究所的學業和論文。

感謝家人對我的支持和鼓勵。尤其是為了我的事比我還要煩惱的父母親，我想說的是，謝謝你們，我畢業了。

也感謝我以前的同學們，能不吝於給我意見和激勵，以及在課業論文、之後當兵和未來工作的經驗分享。

最後謝謝所有曾經在球場上陪伴我發洩壓力的一群球友們。還有幾位這段時間分享心事和一起拼畢業的好友們。

摘要

隨著電腦運算處理能力地增加，在評價金融衍生性商品時也能達到更有效率且正確的結果；而也因為金融商品之計算價格有時會有時間的急迫性，因此如何用電腦硬體和演算法增進計算速度和正確性就成為重要的課題。

在本篇論文中我們使用巴黎選擇權和圖形處理器來做例子，巴黎選擇權為一種擁有障礙選擇權特性的一種路徑相關選擇權。Li 和 Zhao 提出利用生成函數來計算巴黎選擇權的價格，我們在中央處理器和圖形處理器上實作出該方法。結果表示由於圖形處理器有著強大的平行運算能力，因此在執行時間上比中央處理器少上許多，尤其當期數值越大越明顯，而最後價格和中央處理器運算後的價格幾無差異。也因此我們可以更快速地獲得巴黎選擇權的價格而不失其精確度。

關鍵字：巴黎選擇權，障礙選擇權，選擇權評價，圖形運算單元，計算整合裝置結構，平行運算，生成函數。

Abstract

As computing power increases, we can get faster and more correct results in pricing derivatives. How to improve speed and correctness by computer hardware and algorithms is an important issue because pricing financial products is often a time-critical task.

In this thesis we use Parisian options and GPUs as an example. Parisian options are path-dependent options with barrier-like features. Binb-Qing Li and Hai-Jian Zhao proposed to price Parisian options by generating functions. We implement this method in both CPUs and GPUs. The results show that the execution time used by GPUs is much smaller than those by CPUs because of their powerful parallel-processing capabilities, especially when number of periods grows bigger. As a result, we can price Parisian options faster and with accuracy.

Keywords: Parisian options, barrier options, option pricing, GPU, CUDA, parallel processing, generating function.

目錄

圖目錄

表目錄

第一章 選擇權簡介

1.1 選擇權

選擇權為一種權利契約，買方支付權利金後，便有權利在未來約定的某特定日期(到期日)，依約定之履約價格(Strike Price)，買入或賣出一定數量的約定標的物。選擇權分為買權和賣權，執行的動作分為買進及賣出，以買進買權為例，買入買權意味著在到期日時，契約擁有者有權利依履約價格買進契約數量的標的物或是選擇不執行，而當買進買權的人選擇執行契約時，賣出買權的人有義務履約，買入買權者是對市場的未來走勢看漲，所以希望能在未來用較市場低的價格買入標的資產以獲利。

選擇權依買方得要求履約之期限，又可分為「美式」與「歐式」選擇權，美式選擇權的買方能於選擇權到期前任一天執行權利，歐式選擇權的買方只能在到期日才能行使權利[4]。

選擇權的特性在於：

1. 槓桿操作

選擇權的買方只需支付小額權利金，卻有無限獲利的可能，所以有以小搏大，用較低本金獲取較高投資報酬的特性。

2. 避險

投資者若持有現貨，如果不確定市場未來走向，為了規避風險，可以用購買選擇權的方式。好處在於或現貨價值下跌，則選擇權獲利可以彌補現貨損失，而如果現貨價值上漲，則只損失小部份權利金。

3. 遞延投資決策

由於選擇權從生效日到執行日有段時間，因此買方可以有較多的時間觀察市場的走向以及判斷，而美式選擇權更因為於到期日前皆可執行，因此對於買方的資金調度提供更高的彈性。

選擇權又稱為期權。

1.2 障礙選擇權

障礙式選擇權與一般選擇權的最大差異在於：障礙選擇權除了有一般之履約價格外，尚有設計一特定的障礙價格(價格上限或下限)，當標的物價格在契約到期前碰觸到此障礙價格，選擇權契約即立刻生效或終止。其分類如下表所示：

	生效型	終止型
上限型	上限生效型買權 上限生效型賣權	上限終止型買權 上限終止型賣權
下限型	下限生效型買權 下限生效型賣權	下限終止型買權 下限終止型賣權

表 1-1. 障礙選擇權之分類。

以上限終止型選擇權(up-and-out)為例，此選擇權有個大於初始價格的障礙價格(barrier，或稱界限)，當標的物價格達到此障礙價格時，此選擇權宣告失效。而上限生效型選擇權則是當標的物價格觸碰到障礙價格時才開始生效[3]。

1.3 巴黎選擇權

當障礙選擇權的生效(或終止)條件從「觸碰」到障礙價格變成「觸碰且持續一段時間」時，此障礙選擇權便稱為巴黎選擇權[14]。

1.3.1 基本概念

巴黎選擇權可說是一種障礙選擇權的變形，它具備障礙選擇權的特性，價格與標的商品路徑上的價格是否曾觸及障礙價格有關，但是條約規定較為嚴格，可提供投資人之選擇性較多。簡言之，巴黎選擇權為一種價值取決於標的資產是否於某特定期間內碰到或是超過某特定界限，且超過界限的期間長度達到某特定標準的商品。

1.3.2 種類形式

(1)累積型巴黎選擇權：此選擇權價值與標的物價格觸碰或是超過障礙價格

的期間總長度有關。資產標的物價格觸碰或超過障礙價格的「累積」期間長度大於等於我們先預定的長度值時，此選擇權才會生效(或終止)。如同一般障礙選擇一樣，此選擇權共有八種型式。

(2)連續型巴黎選擇權：此種選擇權在資產標的物價格觸碰或超過障礙價格的「連續」期間長度大於等於我們先預定的長度值時，選擇權才會生效(或終止)。如同一般障礙選擇一樣此選擇權共有八種型式。

(3)混合型巴黎選擇權：為以上兩種型式選擇權之混合體，在本篇論文裡不在討論之列。

1.3.3 特性

巴黎選擇權的好處在於他提供了投資人一個權利金較低而且保護較多的一個衍生性商品，尤於它是在標的物價錢碰到或超過某一界限一段時間後即生效或中止，因此權利金價格會比一般普通選擇權低，而也不會如同一般障礙選擇權一樣一但標的物價格觸碰或超過界限馬上就失效，而能提供較多的保護。

而若以累積型和連續型巴黎選擇權做比較，由於連續型巴黎選擇權的條件比較不容易達成，商品比較不容易失效，所以其選擇權價格會比較高。

巴黎選擇權的評價方式除了和一般選擇權一樣的基本參數像是：履約價格，期間長短，無風險利率等，還與下列幾個參數有關：

1. 到期日長短
2. 預定生效(終止)期間長短
3. 障礙價格[14]

第二章 基本觀念以及工具

2.1 二項式選擇權定價模型

二項式選擇權定價模型最早由 Cox，Ross 和 Rubinstein 三人在 1979 年提出，目的在評估到期日前行使的選擇權之合理價值。二項式選擇權定價模型為一種離散時間模型，此外並假設股價只有上升和下跌，而且每次升和跌的機率以及幅度不會改變。而股價變動為一連續時間的變化，但在此模型中將此連續時間切成一樣長度的時段，每個時段都有個觀察點或稱節點，並模擬出所有可能的發展路徑，再對每一個路徑上的每一個節點計算出標的物在該節點時的收益以及選擇權在該節點的價格。

假設現在股價 S，上升和下跌幅度分別為 u 和 d，上升和下跌機率分別為 p 和 q，給定觀察時間 Δt，成交價為 X，選擇權價格為 C (在此假設為買權)，則可以用下圖來表示一期的二項式選擇權定價模型[13]：

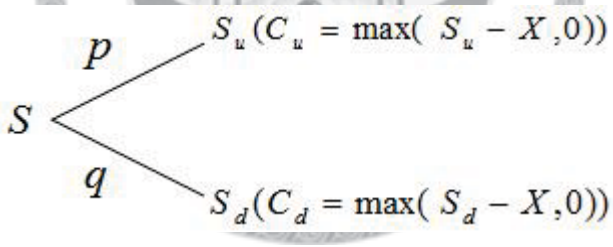

圖 2-1. 一期的二項式選擇權定價模型。

而 C_u，C_d 即為各自所在節點的選擇權價格。

為了方便，我們將 q 設成 $1-p$，假設經過 n 期，則模型變成圖 2-2，假設在某一末端節點中的該點股價經歷 j 期是下跌，$n-j$ 期是上升，則該點之股價為 $Su^{n-j}d^{j}$，到達該點機率為 $\binom{n}{j}p^{n-j}(1-p)^{j}$。將折現率考慮進去的話，我們將可以得出該選擇權的期初價值：

$$c = e^{-rT}\sum_{j=0}^{n}\binom{n}{j}p^{n-j}(1-p)^{j}\max(Su^{n-j}d^{j} - X, 0)$$

其中 r 為無風險利率，T 為選擇權的存續期間。

若將 u 值定為 $e^{\sigma\sqrt{\frac{T}{n}}}$，$d$ 值定為 $\frac{1}{u}$，則此二項式選擇權定價模型即為原始 CRR 模型。

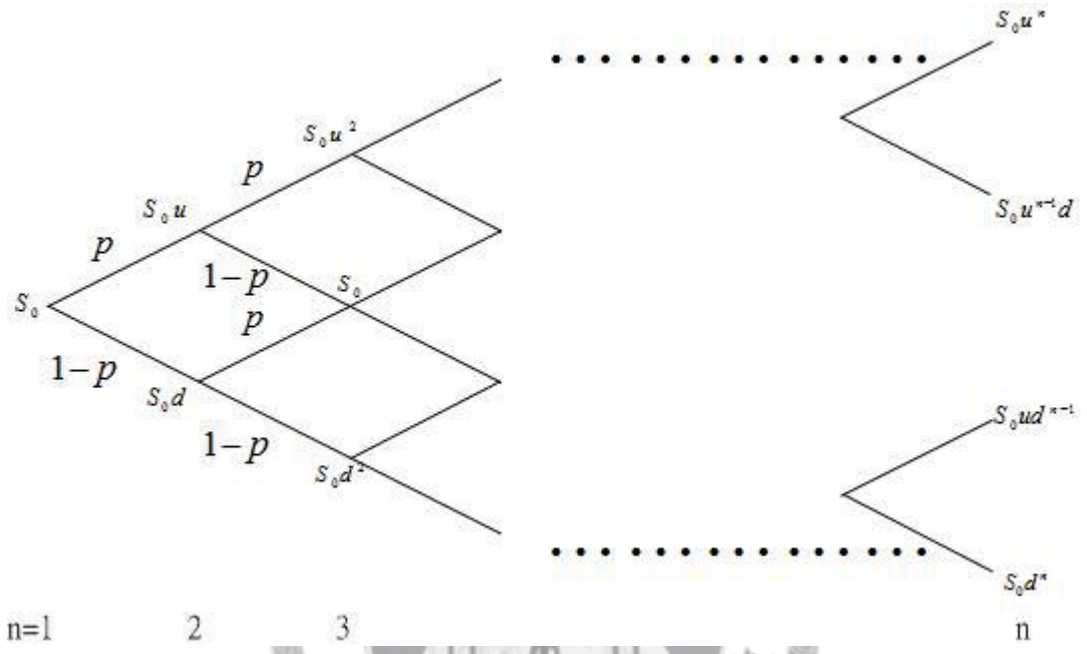

圖 2-2. n 期的二項式選擇權定價模型，S_0 為初始價格。

2.2 生成函數

假設有一數列 $A_n=\{a_0,a_1,\ldots\}$，則函數 $f(x)=\sum_{r=0}^{\infty}a_rx^r=a_0+a_1x+\cdots+a_rx^r+\cdots$ 稱為數列 A_n 之生成函數。

生成函數通常用於解決個數組合之問題。舉例來說，有兩個不同袋子，分別有兩顆球在裡面，則所有可能取球的方法可以用下列式子來表示：(球視為不同)

$$(1+x+x^2)(1+y+y^2)=1+x+x^2+y+xy+x^2y+y^2+xy^2+x^2y^2$$

其中1代表不取，一次方表示取一顆，兩次方表示取兩顆。若球視為相同，則上式可改寫成 $1+2x+3x^2+2x^3+x^4$，x 的次方表示球的個數，係數表示方法數，像是 $3x^2$ 表示取 2 顆球的方法有三種：挑其中一個袋子取其裡面全部的球或是兩個袋子分別取一顆球[2]。

2.3 GPU 和 CUDA

2.3.1 GPU 簡介和歷史

GPU(Graphic Processing Unit)，中文翻成圖形處理器，是一種專門用來處理影像運算工作的微處理器。GPU 最大功用在於使顯示卡減少對 CPU 的依賴，並且分擔了 CPU 的工作，尤其在處理 3D 圖形時效能更加明顯。一顆 CPU 會有一到四顆核心，用來處理序列運算(Serial Processing)，而一張有 GPU 的顯示卡中有多個處理器，相當於 CPU 的運算核心，但不同之處在於 CPU 的核心一次最多只能處理兩條執行緒，而 GPU 的卻總共可以支援上百條以上，所以在平行處理程式平行運算下，使用 GPU 的計算效率會比使用 CPU 快上許多倍。我們日常生活中比較常接觸到平行運算的用途：1.影片與照片、2.視覺網站、3.遊戲。以照片來舉例，一張照片是由多個像素組成，CPU 雖然可以快速地做較複雜地運算，但是由於處理器的限制，CPU 只能一個像素一個相素計算；而 GPU 便可以同時計算多個像素，因此可以同時處理一塊區域的色彩，所以在處理照片上 GPU 顯得有較大地優勢。而我們將此技術運用在選擇權的計價上，因為在二項式模型中，若 n 值(期數)較大，則總路徑數將會達到一個可觀的數字，若用傳統的 CPU 做運算將會耗資大量的時間，因此我們將眼光放在 GPU 上，期望能藉由它的平行運算能力來減少消耗的時間。同樣地，大多數應用程式開發者在做 GPU 運算時，是將程式的連續部份交由 CPU 做，而重覆性高的工作交由 GPU 處理。如此之外，GPU 在處理浮點數運算方面的成長速度也遠大於 CPU，如圖 2-3。但相對的有些晶片並無分開地整數運算單元因此在整數運算方面效率稍差[12]。

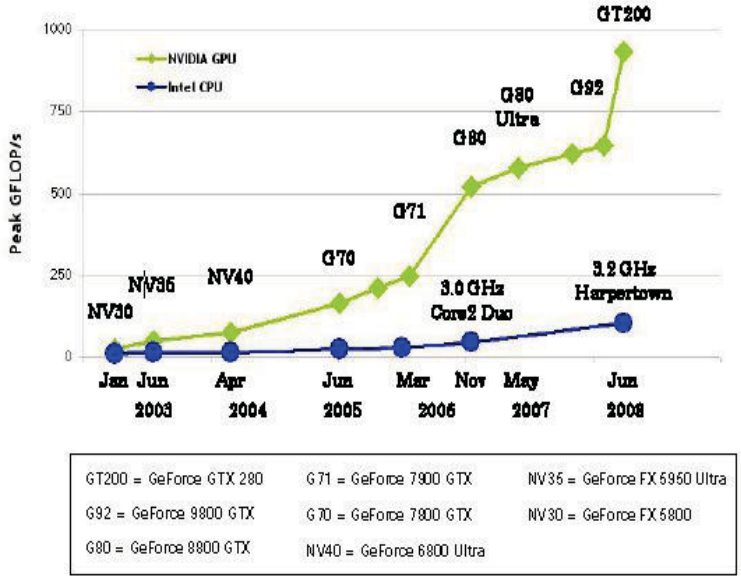

圖 2-3. CPU 和 GPU 分別在浮點數運算的速度 [8]。

而做 GPU 核心最有名的公司當數 NVIDIA，最早於 1998 年發佈 NV4，用於 Riva TNT 顯示卡上；隔年 4 月又推出 NV5，用於 Riva TNT5 上；在 1999 年的 9 月則有了 NV10 的問世，NV10 又稱 TNT3，而 GPU 的概念也就是從這時候開始有的。在 2000 年 4 月又出了 NV15，裝在 NVIDIA 新系列 GeForce 2 GTS 上，其中 TS 代表的是「GigaTexture Shader」，意思是一秒可以填充的像素個數是 10 億級的。在之後又陸續推出了 NV10，用在 GeForce 2 MX 上；NV20，為 GeForce 3 系列顯卡的核心並且將著色單元首次分成像素和頂點兩部份，支 DirectX8；2002 年 2 月同時發佈了 NV17 和 NV25，前者用於 GeForce 4 MX440(為近年來大多數遊戲的最低要求顯示卡配備)，後者用於較高階的型號 GeForce 4 Ti4600。2004 年研發出 NV40 成為 GeForce 6 系列，同年 10 月其競爭對手 ATI 也推出 Radeon 9700，為全球第一個支援 Direct 9.0 的顯示卡。其後 NVIDIA 於 2005 年將 NV 改名為 G 系列，研發出 G70，G72 等等，一直到 2006 年的 G80，用於 GeForce 8800 系列，同時也支援 DirectX 10；以及最近在 2008 年推出的 G200[10]。

2.3.2 GPGPU 和 CUDA

說到 GPU 運算就一定要提到 CUDA，CUDA 全名為 Compute Unified Device

Architecture，為 NVIDIA 公司推出的一個 GPU 整合技術，透過此技術，使用者可以利用 NVIDIA 公司出的 GeForece 8 系列以後的 GPU 來進行運算，也是該公司對於 GPGPU 的正式名稱，所謂 GPGPU 中文全名為 general-purpose computing on graphic processing units，是一種利用 GPU 來計算原本由 CPU 處理的通用計算任務，也就是將 GPU 用在非傳統的 3D 圖形顯示[9]。傳統的 GPGPU 開發方法都是透過 OpenGL 或 Direct3D，以編寫 shading language 的方法控制 shader 來想辦法運算。而 NVIDIA 提出的 CUDA 是可以透過 C 語言的函式庫來編寫的，加上 不使用圖形函式庫，因此在程式設計上更為方便。CUDA 大致上分為 library、runtime、driver 三個部份。其架構圖如圖 2-4：

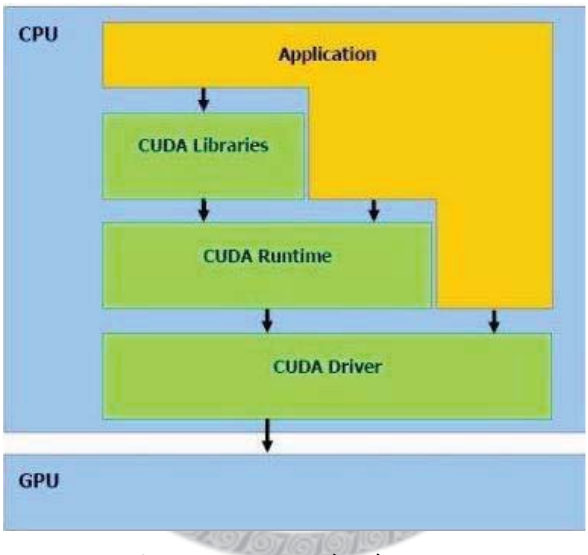

圖 2-4. CUDA 架構 [8]。

而在寫 CUDA 的程式時，我們會將程式執行的區域分成兩個部份，一個是 CPU 執行的 host 端，另外一部份就是 GPU 執行的 device 端。而在 CUDA 程式架構裡，主要的程式還是 CPU 所執行的，只有在遇到需要平行處理的時候，才會將程式編譯成 device 能執行的程式再丟給 GPU 執行，而此程式在 CUDA 中稱之為 kernel。而在 device 中，處理 kernel 的最小單元叫做 thread，中文為執行緒，每一個 device 中有多個 thread，每個 thread 都是同時執行 kernel 程式，而我們所利用的就是每個 thread 的 index 不同，而有不同的資料來進行運算。數個 thread 可以組成一個 block，同個 block 裡的 thread 可以存取同一塊記憶體，因此可以

進行快速的同步動作，而不同 block 的 thread 則無法直接存取同個記憶體，多個 block 可以組成一個 grid，透過這種模式，我們可以更加有效地利用每個 thread 的功效，而不會被 thread 數目所限制。其關係如圖 2-5。

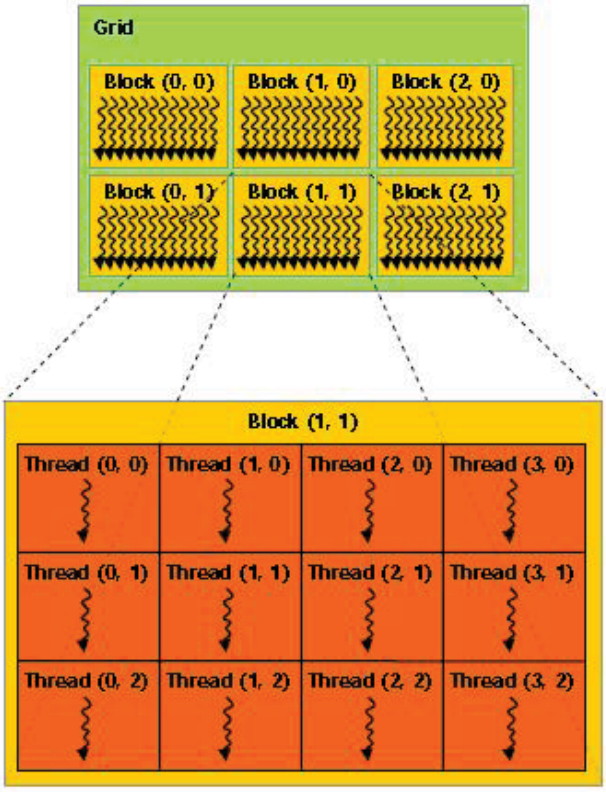

圖 2-5. Thread、Block 和 Grid 的關係 [8]。

在寫 CUDA 程式時，要讓 thread 所使用的資料都會先從 host 傳進 device 的記憶體中，而根據不同的應用又有分不同的記憶體類型。如圖 2-6 所示，每個 thread 有自己的 local memory，而在同一個 block 裡的不同 thread 則可存取同一個 shared memory，而所有的 thread 即使在不同 block 甚至不同 grid，則都能使用同一個唯讀的記憶體也就是 global memory 裡的資料。除此之外，還有兩種唯讀的記憶體空間，分別稱為 constant memory 和 texture memory，也是能讓所有 thread 所存取的空間。上述的記憶中，global、constant 和 texture 記憶體的存續時間是和 kernel 程式存在時間一樣長的。而我們常要注意的是 thread 在存取同一塊記憶體時的同步化。

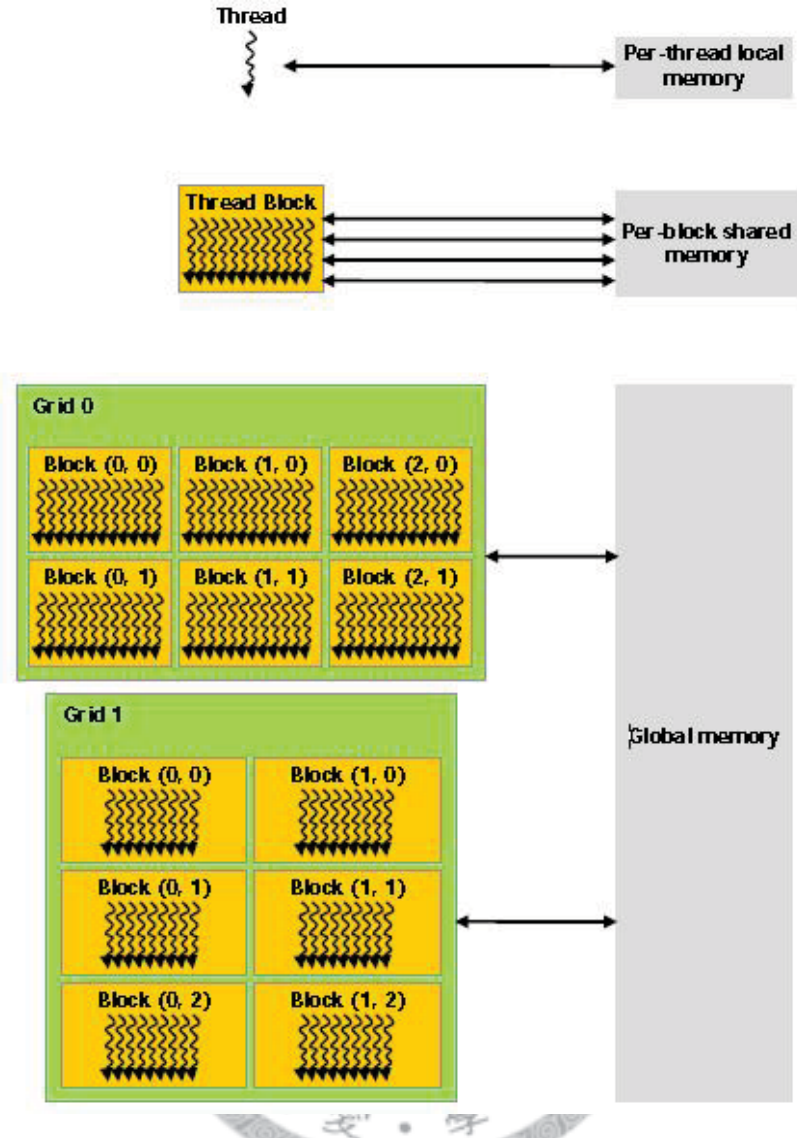

圖 2-6. 記憶體層次 [8]。

而 CUDA 程式執行時的流程為：

1. Host(通常為主機)記憶體傳送資料和程式碼到 device(通常為顯示卡)記憶體。
2. Host 做其他的事或閒置，device 執行 kernel 程式。
3. Device 記憶體將執行結果傳回 host 記憶，程式繼續執行。

第三章 利用生成函數來計算巴黎選擇權的價格

在這個章節中，我們則實作 Li 和 Zhao 所提出用生成函數來評價巴黎選擇權的方法，一開始先是以 CRR 模型來做一些定義和介紹，接著便用求出生成函數的係數方法來解出巴黎選擇權在 n 期時的價格[1]。

3.1 組合數學架構

在 CRR 模型中，我們是用 lattice path 做基礎，並假設股價只有上漲和下跌兩種可能，我們將每一個上漲或下跌一期的路徑稱為單元路徑。而如果在座標圖上，每一個上漲的單元路徑就是從 (x, y) 移動到 $(x+1, y+1)$，我們用 U 來表示。每個下跌的單元路徑就是從 (x, y) 移動到 $(x+1, y-1)$，我們用 D 來表示。所以在此模型中，每條 lattice path 我們都可以改寫成 $a_1 a_2 \cdots a_n$，其中 $a_i \in A = \{\text{U, D}\}$，而且此路徑長為 n。此外我們還定義空路徑為 1。而兩條路徑的乘積我們視為連結，假設一條路徑 $\alpha = a_1 a_2 \cdots a_n$，另一條 $\beta = b_1 b_2 \cdots b_m$，則他們的乘積即為 $\alpha\beta = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$，也是一條新的路徑。

接著我們為了找出路徑為 n 的所有 lattice path 個數，便引進了生成函數的概念，假設 L 為所有 CRR 模型中所有 lattice path 的集合，我們定義 L_n 為在所有在 L 中而且長度等於 n 的路徑集合，其數學式為：$L_n = \{\alpha \in L : \ell(\alpha) = n\}$。再假設 f_n 為 L_n 的元素個數，用 $|L_n|$ 做表示，則我們可以寫出序列 $\{f_n\}$ 的生成函數：$L(t) = \sum_{n \geq 0} |L_n| t^n$，而又因為 $L = \{\alpha \in L_n : n \geq 0\}$，因此上面 $\{f_n\}$ 的生成函數也可以視為 L 集合的生成函數。接下來我們定義 $U = \{\text{U}\}$，也就是說在 U 集合裡只有一個元素叫做 U，很明顯可以看出 U_n 中除了 $U_1 = U$ 以外其他皆為空集合，因此 U 集合的生成函數為 $U(t) = t$，同理來說，若我們定義 $D = \{\text{D}\}$ 則 $D(t) = t$，我們也可以將 t 視為在 lattice path 中的一步，不是往上就是往下。

這時以一個下限終止型障礙選擇權買權來作例子，首先我們先將 CRR 模型標在座標圖上，而每條與 x 軸平行且和 x 軸距離為 k 的線稱為第 k 層，第 0 層即

為 x 軸。我們可以想縱軸為標的物價格，橫軸為時間，假設此選擇權從原點 $(0,0)$ 開始，而障礙價格為第 -1 層，也就是說當某條價格路徑碰到第 -1 層時則此選擇權失效，因此當我們要計算此選擇權價值時，需要考慮的是所有未觸碰到第 -1 層的路徑，每條路徑終點為 (n,i)，其中 $-n \le i \le n$。為了簡化計算，我們這邊只需考慮所有未觸碰到第 -1 層且終點在 $(2n,0)$ 的路徑。

假設一集合 C 為所有未觸碰到第 -1 層且終點在 x 軸上的 lattice path，C_n 為 C 裡且終點在 $(2n,0)$ 的總路徑個數，則 C 的生成函數為 $C(t)=\sum_{n\ge 0} C_n t^{2n}$，而在計算 $C(t)$ 的公式前，我們先介紹質數集合：在一 lattice path 所集合而成的集合 L 裡，若 L 裡面每一條路徑都可以被唯一地分解成數個不同的小路徑，則此種小路徑的所有集合即為集合 L 的質數集合 P。而我們可以使用一個輔助定理為：$L(t)=\dfrac{1}{1-P(t)}$。此外我們還可以知道，在 C 集合裡的質數集合 P 是由所在 C 裡而且只有開始和終點在 x 軸上的路徑所組成，因此我們可以將任何在 P 裡的路徑分解成 UβD，其中 $\beta \in C$，而 P 的生成函數 $P(t)$ 就等於 $U(t)C(t)D(t)$，再利用剛剛的定理，則等式變成：$C(t)=\dfrac{1}{1-t^2C(t)}$，經過化簡和二元一次方程式解根的公式，我們可以導出 $C(t)=\dfrac{1-\sqrt{1-4t^2}}{2t^2}$，再經過泰勒展開式的變換，我們可以得到 t^{2n} 的係數 $C_n=\dfrac{1}{n+1}\binom{2n}{n}$，這也是有名的 Catalan 數。而 C_n 既為所有未經過 x 軸底下且終點在 $(2n,0)$ 的路徑個數，它的值將我們等等要計算巴黎選擇權價格時會用到。

除此之外，我們用 Lagrange 反算公式可以求出 t^n 在 $(C(t))^r$ 時的係數為

$$\frac{2r}{n+2r}\binom{n+r-1}{\frac{n}{2}}, r \in N^+$$

在我們要開始計算連續型巴黎選擇權價格前，還要介紹一個重要的定理：

[定理一]

令 m 為一整數，選一個路徑 μ，定義「μ 在第 m 層上最多的連續步數」為 $mc_m(\mu)$，今天給定一個正整數 l，則所有 $mc_m(\mu)<l$ 並且碰到或越過第 $m-1$ 層以及終點在第 i 層的路徑 μ 所成的集合為 T，其中 $i<m$，則 T 的生成函數為：

$$T(t)=\frac{C(t)-Q(t)}{t^2Q(t)^2-Q(t)+1}t^rC(t)^r,$$
$$r=2m-i-2,$$
$$Q(t)=C_0+C_1t^2+\cdots+C_{\frac{l-2}{2}}t^{l-2}$$

下面的輔助定理則是在我們計算累積型巴黎選擇權時會用到。

Chung-Feller Theorem：從 (0,0) 到 (2n,0) 並且由單位路徑 U、D 所組成，並且正好有 2k 步在第 0 層上的所有路徑，其個數和 k 值無關，而是和第 n 個 Catalan 數一樣。

3.2 計算巴黎選擇權的價格

在這章我們所要計算的是 n 期上限終止型巴黎選擇權買權的價格，在計算之前先對一些數值做定義，假設我們的障礙價格 H 為 S_0u^m，m 為股價碰到障礙價格時要走的最少向上次數，w 為我們設定的區間，也就是如果股價在障礙價格上連續存在 w 時間後此選擇權失效，l 為相對應 w 的，在障礙價格上的時間點步數，$l\geq\frac{w}{T}n$，我們通常將 l 假設成偶數。

3.2.1 連續型巴黎選擇權

連續型巴黎選擇權只考慮在障礙價格上連續存在的的時間，因此我們用剛剛定義的「μ 在第 m 層上最多的連續步數」也就是 $mc_m(\mu)$，並且再定義 $f(n,i)$ 為從原點 (0,0) 開始，終點在 (n,i) 且 $mc_m(\mu)<l$ 的路徑 μ 之個數。很明顯當 $n+i$ 為奇數時，$f(n,i)=0$，因此我們直接將 $n+i$ 作為偶數。而根據 i 值不同，分成三個情況討論：

第一種情況，$i\geq l+m$：在這情況中，$f(n,i)=0$。因為當此路徑碰到第 m 層後，它還須要至少 $i-m\geq l$ 個向上步數才達到第 i 層。

第二種情況，$i<m$：在這個情況我們又要再依據這條路徑是否有碰到第$m-1$層來分成兩部份討論。假設$g(n,i)$為所有$mc_m(\mu)<l$而且曾經碰到或越過第$m-1$層且終點在(n,i)的所有路徑μ數。然後用剛剛的定理一，我們可以獲得以下的生成函數:

$$\sum g(n,i)t^n = \frac{C(t)-Q(t)}{t^2Q(t)^2-Q(t)+1}t^{r_1}C(t)^{r_1},$$
$$r_1 = 2m-i-2,$$
$$Q(t) = C_0 + C_1t^2 + \cdots + C_{\frac{l-2}{2}}t^{l-2}$$

再用之前提到的t^n在$(C(t))^r$時的係數公式以及運算後，可以得出t^n的係數為：

$$\frac{2r_1+2}{n+l+r_1+2}\binom{n+l}{\frac{n+l-r_1}{2}} - \sum_{t_1\le j\le t_2}\frac{r_1}{j+r_1}\binom{2j+r_1-1}{j}C_{\frac{n+l-r_1}{2}j}$$

其中$t_1 = \frac{n-r_1+2}{2}, t_2 = \frac{n+l-r_1}{2}$。

再設$h(n,i)$為從未碰到第$m-1$層的路徑數，根據反射原理，我們可以直接推導出

$$h(n,i) = \binom{n}{\frac{n+i}{2}} - \binom{n}{\frac{n+i}{2}-(m-1)}$$

最後$g(n,i)$和$h(n,i)$兩者相加即為$f(n,i)$

第三種情況，$m\le i<l+m$：每個路徑都可以分成兩個部份，第一個部分是從原點到$(n-j,m-1)$，其中$i-m+1\le j\le l$，這部份的路徑數量為$g(n-j,m-1)$。第二部份的路徑長度為j，$j\le l$，可以想成從原點出發，終點在$i-(m-1)$而且途中沒有再回到原點的路徑。而其生成函數可寫為$(tC(t))^{i-(m-1)}$。再用之前寫過的t^n在$(C(t))^r$時的係數公式，則係數即為這部份路徑數量，計算後為：

$$\frac{2r_2}{j+r_2}\binom{j-1}{\frac{j-r_2}{2}}, r_2 = i-(m-1), r_2\le j\le l$$

最後我們再將兩部份做乘積就可得出當 $m \le i < l+m$ 且 $mc_m(\mu) < l$ 時的所有路徑 μ 之總個數：

$$f(n,i) = \sum_{i-m+1 \le j \le l} g(n-j, m-1) \cdot \frac{2r_2}{j+r_2} \binom{j-1}{\frac{j-r_2}{2}},$$
$$r_2 = i-(m-1)$$

而在我們做最後的加總前，為了節省運算成本，我們先算出一個最多往下步數 a 使得低於此數的選擇權最後都會在價內：$a = \left\lfloor \frac{\log(S_0 u^n / K)}{\log(u/d)} \right\rfloor$

最後，我們可以導出上限終止型的連續型巴黎選擇權的價格：

$$c = e^{-rT} \sum_{j=0}^{a} f(n, n-2j) p^{n-j} (1-p)^j (S_0 u^{n-j} d^j - K)$$

其中 $f(n,n-2)$ 為 $n-j$ 期上漲 j 期下跌且 $mc_m(\mu) < l$ 的路徑數。

3.2.2 累積型巴黎選擇權

累積型巴黎選擇權所考慮的是股價路徑在障礙價格上累計的時間總數，如同 $mc_m(\mu)$，我們定義「μ 在第 m 層上的累積步數」為 $cs_m(\mu)$，而 $f_l(n,i)$ 為從原點 $(0,0)$ 開始，終點在 (n,i) 且 $cs_m(\mu) < l$ 的路徑 μ 之個數。$n+i$ 亦作為偶數。而也根據 i 值分成三個情況討論：

第一種情況，$i \ge l+m$：和連續型巴黎選擇權一樣，$f_l(n,i) = 0$。

第二種情況，$i \le m$：根據路徑有沒有越過第 m 層分成兩部份討論，假設 R 為所有路徑 μ 且 $cs_m(\mu) < l$ 並且曾越過第 m 層而最後終點在 i 的集合，則每個 R 裡的 μ 都可分解成三個部份 αβγ，第一部份 α 為所有從原點開始只碰到第 m 層一次也就是終點在第 m 層的的路徑，其集合稱為 R_1；第二部份 β 為所有 $cs_0(\beta) < l$ 而且在第 0 層上最少兩步且終點在上面的路徑，其集合稱為 R_2；第三部份 γ 為從第 0 層出發，終點在第 $i-m$ 層且途中沒有再回到第 0 層的路徑，其集合稱為 R_3。而我們可以求出 R 的生成函數：$R(t) = R_1(t)R_2(t)R_3(t)$，其中 $R_1(t) = (tC(t))^m$，而 $R_3(t) = (tC(t))^{m-i}$。接著我們要算 $R_2(t)$，利用剛剛介紹的 Chung-Feller 定理，我

們假設 A_n^k 為終點在 $(n,0)$ 而且正好有 k 步在第 0 層上的路徑數量，則 $A_n^k = C_{\frac{n}{2}}$，我們則能寫出下式：

$$\begin{aligned} R_2(t) &= A_2^2 t^2 + \cdots + (A_{l-2}^2 + \cdots + A_{l-2}^{l-2})t^{l-2} + \sum_{n>l-2}(A_n^2 + \cdots + A_n^{l-2})t^n \\ &= C_1 t^2 + \cdots + \frac{l-2}{2} C_{\frac{l-2}{2}} t^{l-2} + \frac{l-2}{2} \sum_{n>l-2} C_{\frac{n}{2}} t^n \\ &= \frac{l-2}{2} C(t) - \sum_{j=0}^{l-4} \frac{l-2-j}{2} C_{\frac{j}{2}} t^j \end{aligned}$$

設 $g_l(n,i)$ 為在 R 中終點在 (n,i) 的所有路徑數，則我們根據上面能得到 R 的生成函數為：

$$\sum g_l(n,i)t^n = \frac{l-2}{2} t^{2m-i} C(t)^{2m-i+1} - (tC(t))^{2m-i} \sum_{j=0}^{l-4} \frac{l-2-j}{2} C_{\frac{j}{2}} t^j$$

再假設 $r_3 = 2m-i$，然後代入 t^n 在 $(C(t))^r$ 時的係數公式，可以導出：

$$g_l(n,i) = \frac{(l-2)(r_3+1)}{n+r_3+2} \binom{n}{\frac{n-r_3}{2}} - \sum_{j=0}^{l-4} \frac{r_3(l-2-j)}{n-j-r_3} \binom{n-j-1}{\frac{n-j-r_3}{2}} C_{\frac{j}{2}}$$

設 $h_l(n,i)$ 為從未碰到第 m 層的路徑，同樣地也不會碰到第 $m+1$ 層，因此可以求出：

$$h_l(n,i) = \binom{n}{\frac{n+i}{2}} - \binom{n}{\frac{n+i}{2} - (m+1)}$$

最後我們可以將兩部份加起來求出 $f_l(n,i) = g_l(n,i) + h_l(n,i)$

第三種情況，$m < i < l+m$：這裡的 $f_l(n,i)$ 和剛剛在算連續型巴黎選擇權的第三中情況類似，可用下式表示：

$$f_l(n,i) = \sum_{i-m+1 \le j < l} f_{l-j}(n-j,m) \frac{2r_4}{j+r_4} \binom{j-1}{\frac{j-r_4}{2}},$$
$$r_4 = i-m$$

其中 $f_{l-j}(n-j,m)$ 為路徑 η 其中 $cs_m(\eta) < l-j$ 且終點在 $(n-j,m)$ 的路徑數

量。而剩下從 $(n-j,m)$ 到 (n,i) 且從未碰到第 m 層的路徑數為 $\frac{2r_4}{j+r_4}\binom{j-1}{\frac{j-r_4}{2}}$。

最後，我們可以導出上限終止型的累積型巴黎選擇權的價格：

$$c = e^{-rT}\sum_{j=0}^{a} f_l(n,n-2j)p^{n-j}(1-p)^j(S_0u^{n-j}d^j - K)$$

a 的定義和上面我們在計算連續型巴黎選擇權時的定義的一樣，為一個最多往下步數使得低於此數的選擇權最後都會在價內：$a = \left\lfloor \frac{\log(S_0u^n/K)}{\log(u/d)} \right\rfloor$。

第四章 實作和數據結果討論

我們在實作第三章提到的方法時，是在 Windows 環境下，使用微軟的 Visual C++。而一開始我們先比較我們所要使用的 CPU 和 GPU 硬體環境不同，接著我們大概描述如何用 C 語言來寫主程式以及 CUDA 程式，最後在不同的變數下，來驗證我們的確能利用 GPU 獲得更好的效能。

我們所使用的 CPU 和 GPU：

	Intel Core2 Duo T7100	Nvidia GeForce 8400M GS
核心數目	2	16
執行緒數目	2	自訂
核心頻率(速度)	1.8GHz	400MHz

表 4-1. CPU 和 GPU 的比較。

值得注意的是，因為我們寫在 CPU 上的程式並未考慮到雙核心，因此當 CPU 在處理程式時是用一顆核心以及一條執行緒去執行的。

在開始實驗前，我們知道當期數變多時，要計算標的物價格的路徑總數的數量增加是龐大的，甚至超過 C 語言裡 double 變數所能容納的最大值，因此我們在這使用了 Lyuu 和 Wu 所寫的論文[6]裡的方法，將數字取了對數之後再計算。

我們設的參數值為：

$$S=100, K=95, H=110, r=0.08, \sigma=0.2, T=1, w=15 \text{ (days)}$$

其中 S 為標地物的初始價格，K 為履約價格，H 為障礙價格，r 為年度無風險利率，σ 為波動度，T 為選擇權存續期間(以年來算)，w 為能在障礙價格上最多不會讓此選擇權失效的天數。

我們先在 CPU 上執行程式，其中期數 n 值我們設 100，200,…,1000，並計算出時間以及結果。接下來我們換成用 GPU 去處理，而除了期數以外，我們還會

設定 GPU 執行緒(thread)的數目，試圖除了證明 GPU 的效能外還看能否找出最佳化的方法。

首先我們先算出兩邊計算價格的結果並且和 Li 以及 Zhao 在他們所計算出的結果做比較。

期數	Li 和 Zhao 的結果	CPU 的結果	GPU 的結果
100	1.1684	1.167366	1.167224

表 4-2. 比較 Li 和 Zhao 以及我們在 CPU 和 GPU 上實作計算連續型巴黎選擇權的價格的結果。

期數	Li 和 Zhao 的結果	CPU 的結果	GPU 的結果
100	1.0157	1.016322	1.015997

表 4-3. 比較 Li 和 Zhao 以及我們在 CPU 和 GPU 上實作計算累積型巴黎選擇權的價格的結果。

在這邊我們看到兩邊結果是差不多的，可能的誤差原因之一為我們取對數後做運算再轉回去時各會有些誤差。另一原因在於雖然 CUDA 現今已經提供雙精確浮點數(double)運算，但還是必須看硬體能支援的計算能力版本(Compute Capability)，像我們在這篇論文所使用的 Nvidia GeForce 8400M GS 這張顯示卡就只能支援到 1.1，也因此無法用雙精確浮點數來算，而必須用單精確浮點數，因此會有些誤差。

接下來我們在 GPU 上設定不同數目的執行緒以及不同期數來計算執行時間。在這邊我們要注意的是，由於在 Li 和 Zhao 的結果中，期數皆是 100 的倍數，因此我們將 thread 也就是執行緒數目設為 100，而因為硬體限制問題，在這張顯示卡上我們最多能設定的執行緒數目為 128。

當執行緒數目為 1 時：

期數	CPU 處理時間(ms)	GPU 處理時間(ms)
10	0.030171	0.046863
20	0.108813	0.400006
50	0.850527	12.498731
100	6.545664	73.193116
200	40.502705	396.019381

表 4-4. 在 執行緒為 1 時，CPU 和 GPU 上計算連續型巴黎選擇權的價格的時間比較。

期數	CPU 處理時間(ms)	GPU 處理時間(ms)
10	0.019905	0.024478
20	0.025283	0.352692
50	0.090444	1.103192
100	0.433924	6.312448
200	2.361334	40.017239

表 4-5. 在 執行緒為 1 時，CPU 和 GPU 上計算累積型巴黎選擇權的價格的時間比較。

在這邊我們可以看出，當執行緒數目為 1 時，GPU 在處理此演算法方面的效率不如 CPU，尤其是當期數越大時越明顯，以累積型巴黎選擇權為例，如表 4-5，當期數為 10 時，GPU 處理所花的時間為 CPU 之 1.55 倍，到了期數為 20 時則為 3.67 倍，隨著期數增加所花費的時間也以更高的倍率增加，當期數為 200 時，GPU 所花的時間已經差不多是 CPU 的 9.77 倍，相當於效率為 0.102 倍，這

是由於我們沒有用到 GPU 最大的功能也就是平行化所致。

當執行緒數目為 100 時：

期數	CPU 處理時間(ms)	GPU 處理時間(ms)
10	0.044140	0.035817
100	5.995385	2.331474
200	47.229465	8.078661
500	648.958618	44.038397
1000	7926.140625	113.346356

表 4-6. 在 執行緒為 100 時，CPU 和 GPU 上計算連續型巴黎選擇權的價格的時間比較。

期數	CPU 處理時間(ms)	GPU 處理時間(ms)
10	0.013130	0.011041
100	0.423657	0.174266
200	2.388921	0.493527
500	33.134659	2.848529
1000	424.776459	7.332075

表 4-7. 在 執行緒為 100 時，CPU 和 GPU 上計算累積型巴黎選擇權的價格的時間比較。

接著我們將執行緒數目設為 100，由表 4-6，4-7 能看出很明顯地在多了平行化之後的改善。以連續型巴黎選擇權來說，在期數為 10 時，GPU 雖然速度較快但不明顯，這是因為當期數太小時，迴圈要重覆執行的次數較少，而 CPU 核心的高處理速度減少了因為增加處理次數所以多花費的時間，加上 GPU 還要處理

資料傳遞問題，因此改善效益不明顯，而在期數增加到 100 後，則開始有較好的改善效益，已經有 2.571 倍的加速。在期數為 500 時，GPU 的執行速度已經是 CPU 執行速度的 14.74 倍。而期數到 1000 後，兩邊執行時間的差異已經有 69 倍多，在累積型選擇權方面，CPU 所花的時間則為 GPU 所花費時間的 57 倍之多。

第五章 結論

巴黎選擇權評價是不少經濟學家或數學學家研究的選擇權，而 Li 和 Zhao 所提出用生成函數來計算巴黎選擇權又提供了一個解決方法，然而，在此演算法中，要計算出選擇權價格所需要的時間量在期數增加時也會大幅地增加，因此，我們便想利用 GPU 的平行化能力來增進效能。

由實驗結果看出，當 GPU 執行緒的數目為 1 時，GPU 執行的環境和 CPU 一樣。但 GPU 執行速度較 CPU 慢，即使此巴黎選擇權計算過程並不複雜而 GPU 處理浮點數運算的效能又比 CPU 好，當期數增加時，GPU 在處理的時間上仍然是多於 CPU。

而當我們設定 GPU 執行緒的數目為 100 時，很明顯能看出擁有平行化處理後，GPU 的效能優於只有單一執行緒的 CPU，但因為我們在交給 GPU 執行時，還要處理資料傳遞，記憶體等問題而消耗一些時間。所以當期數很小時，GPU 平行化的優勢並不明顯。但等到期數大幅增加時，我們可以看到 GPU 執行所花的時間增加的不多，反而是 CPU 執行時間增加幅度變大。這是由於當我們將期數值增大時，執行緒數目為 100 的話每個執行緒只要多跑幾次，而單一執行緒則要處理所有的計算過程所至。

因此，即使現今在使用 CUDA 上仍有一些缺點，比如說被硬體(顯示卡)所限制住，以及無法處理遞迴等，我們仍然能看出 GPU 在處理類似問題的優勢，而除了在本篇演算法以外，GPU 在處理蒙地卡羅模擬，或是一些可以將擁有龐大數量的計算部份拆解成許多彼此相關性不大的計算部份問題等方面也是有著相當的效益。之後隨著硬體技術的進步，我們在處理大量資料的效率改善會更加明顯。

參考文獻

[1] Bing-Qing Li , Hai-Jian Zhao, Pricing Parisian Options by Generating Functions, *The Journal of Derivatives*, 2009, 72–81.

[2] H. S. Wilf, *Generatingfunctionology*, Academic Press, 2nd edition, 1994.

[3] Costabile, M., A combinatorial approach for pricing Parisian options, *Decisions in Economics and Finance*, 2002, 25(2), 111–125.

[4] Hull, J.C., *Options, Futures, and Other Derivatives*, 6th edition, Upper Saddle River, NJ: Prentice-Hall, 2006.

[5] Yuh-Dauh Lyuu and Yi-Chun Wu, Performance of GPU for a Tree Model for Convertible Bonds Pricing with Stock Price, Interest Rate, and Default Risks, 2008.

[6] Yuh-Dauh Lyuu and Cheng-Wei Wu, Pricing Parisian Options: Combinatorics, Simulation, and Parallel Processing, 2008.

[7] Yuh-Dauh Lyuu and Cheng-Wei Wu, An Improved Combinatorial Approach for Pricing Parisian Options, *Decisions in Economics and Finance*, 33(2010) , 49–61.

[8] NVIDIA Corporation, *NVIDIA_CUDA_Programming_Guide_2.2.1*, 2009.

[9] Heresy' Space, http://heresy.spaces.live.com/blog/.

[10] NVIDIA GPU 核心歷史回顧展, http://www.fevernet.com/thread-6501-1-1.html.

[11] CUDA ZONE, http://www.nvidia.com.tw/object/cuda_home_new_tw.html.

[12] CUDA Wiki, http://zh.wikipedia.org/zh-tw/CUDA.

[13] 陳文魁, 選擇權二項式訂價.

[14] 童雅靖, 巴黎選擇權介紹, *寶來金融創新季刊第二十五期*, 2003.