行政院國家科學委員會補助專題研究計畫 □ 成 果 報 告
■期中進度報告

（計畫名稱）

基於圖形處理器之鉅量資料分析系統

計畫類別：■ 個別型計畫　□ 整合型計畫
計畫編號：NSC 98－221－E－009－074－MY2
M執行期間：　98 年 8 月 1 日至 100 年 7 月 31 日

計畫主持人：袁賢銘
共同主持人：
計畫參與人員： 邱俊傑、羅國亨、許文峰

成果報告類型(依經費核定清單規定繳交)：■精簡報告 □完整報告

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：

中　華　民　國　99 年　05 月 30 日

目錄

前言

處理大量資料的能力是在許多領域裡相當基本的。從資料探勘、自然語言分析、地震資料分析、人工智慧到計算物理、化學、生物科學，都需要這樣的能力。然而到目前為止，這樣的分析與模擬都僅限於在一些高效能計算中心中，才有可能被有效率的實現；這些計算中心使用了大量的分散式系統，運用數百台機器及高速的網路設備串接，獲得較高的資料處理能力。然而，這種系統的建置與維護的高成本使得這種運算的資源沒辦法讓每一個研究學者或科學家頻繁的使用，迫使他們得在自己的低效能的機器或叢集上進行研究與資料的分析，甚至花費幾天到幾個星期在等待上。這些在計算中心才能被實現的問題通常不是受限於 I/O 就是受限於運算能力，所以沒有辦法有效的在個人電腦上單單一顆 CPU 搭配傳統的硬碟來執行。然而，因為一些最近個人電腦硬體技術上的進步，本計畫希望透過高效能的繪圖晶片與固態硬碟來解決這個問題，以下我們分別簡介這兩者的演進。從 2003 年開始，在高效能運算的領域裡，運用繪圖晶片（GPU）來加速計算，一直是非常熱門的話題。從早期運用 3D 渲染函式庫到現在有標準的平行運算開發框架，繪圖晶片運算的技術可以說是一日千里。許多原本需要耗費數天甚至是數個月的複雜計算，透過繪圖晶片的幫忙，在短短的幾秒到幾分鐘內就產生結果。然而，到目前為止，要實際運用繪圖晶片的運算能力，開發一個能在繪圖晶片上運行的應用並不是那麼容易，不是每一個研究學者都有能力去發展的。

另外，高速的儲存裝置也從 2007 年開始萌芽，運用高速的快閃記憶體取代傳統機械式硬碟，稱為固態硬碟（Solid-state Drive，SSD），大幅度的提升 I/O 的輸出輸入效率，從以前每秒 50~60MB 的讀取速度一舉提升到每秒 160~170MB。而在 2008 年時，SSD 硬碟價格滑落，開始漸漸的普及到一般消費者，所以在未來幾年內，SSD 硬碟將有機會取代傳統機械式硬碟，成為下一世代個人電腦上儲存裝置的標準。

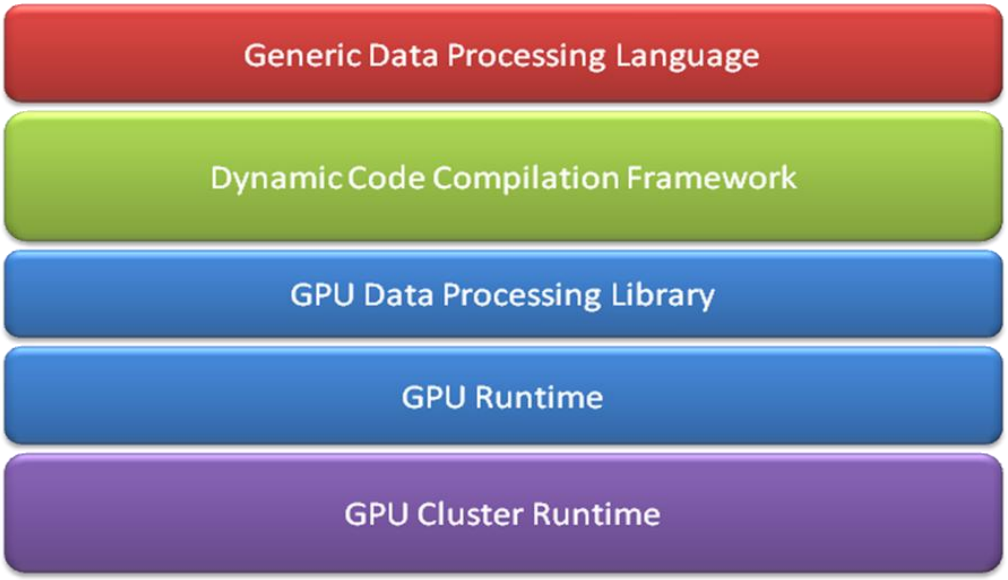
應因繪圖晶片效能的提升及高速運算的需求，以及高速儲存裝置的興起，所以本計畫希望能運用消費者市場中標準的硬體元件，建置一套低成本的高效能運算叢集，並開發此運算叢集上之軟體系統，提供簡單的且專為資料分析設計之語言介面，方便科學家或各個領域的研究學者，運用繪圖晶片上的運算能力及高速的 I/O 系統，在極短的時間內完成他們的資料分析或處理，達到高效能及簡化軟體開發的目標，而不需要了解底層繪圖晶片的架構與高速 I/O 系統的設計。

研究目的

本計畫希望結合高速的 I/O 裝置以及繪圖晶片的運算能力，設計一個高效能低成本的資料處理叢集系統，同時提供友善的使用者介面，使用者不用了解背後複雜的系統設計，也不需要了解如何在繪圖晶片上撰寫程式，更不需要自行管理這些可觀的計算及儲存資源，只要使用簡單專為資料處理設計的語言，給定相對應的資料，系統就能快速的將資料處理完成。

研究方法

a. 系統架構



本計畫欲開發的資料處理系統架構圖如上所示。從底層看上來，包含了 Generic Data Processing Library，Dynamic Code Compilation Framework，GPU Data Processing Library，GPU Runtime 以及 GPU Cluster Runtime，我們將先分項簡單說明如下：

b. Generic Data Processing Language & Dynamic Code Compilation Framework

就整個系統而言，我們希望這個系統是非常泛用的，能提供給學界與業界在

各個領域的專業人士，只要有大量資料分析的需求，便能夠使用。所以我們參考了做產業界大量資料處理最著名的公司，也就是 Google，看看他們是如何處理數個 PB（1 PetaBytes = 1000 TeraBytes = 1000000 GigaBytes）的資料。

c. GPU Data Processing Library

為了提供上層 Generic Data Analysis Language 在處理資料時所需要的函式，本系統也將實作一套 GPU Data Processing Library，在繪圖晶片的運算平台上，實作一個基於繪圖晶片之泛用的資料處理函式庫；其中包含基本之統計功能，平行化之資料結構，字串處理及數值運算函數等。

這部份的函式庫我們將整合現有的已移植到繪圖晶片上的函式庫，在 NVIDIA CUDA 上包括 CUBLAS（CUDA Basic Linear Algebra Subprograms）、CUDPP（CUDA Data Parallel Primitive），或是在 AMD Stream 上包括了 ACML（AMD’s Core Math Library）等；另外也會實作一些常用之 2D 或 3D 平行資料結構如 QuadTree 或 Octree，以及一些基本統計的功能。

d. GPU Runtime & GPU Cluster Runtime

GPU Cluster Runtime 提供了一個通訊及執行控管的平台，讓多台配有繪圖晶片的機器能互相溝通合作，結合多台機器的運算及儲存資源，處理更大量的資料。在這邊我們可採用格網的架構或是 MPI 的環境，結合本計畫要開發之對繪圖晶片上行程管理的模組，以及自動資料及工作切割的模組，提供上層 GPU Runtime 一個統一執行的環境。

關於資料的切割及工作排程，我們將考慮每個節點上之運算單元的效能（註：繪圖晶片的速度及數量），等比例的將資料集合做切割，透過分散式之檔案系統，動態分配到各個節點上，再交由上層的 GPU Runtime 進行計算。

而 GPU Runtime 為 NVIDIA 或 AMD 所提供的能在繪圖晶片上執行程式的介面，如 NVIDIA CUDA 以及 AMD Stream。在本計畫中，我們將選擇其中一種平台做為

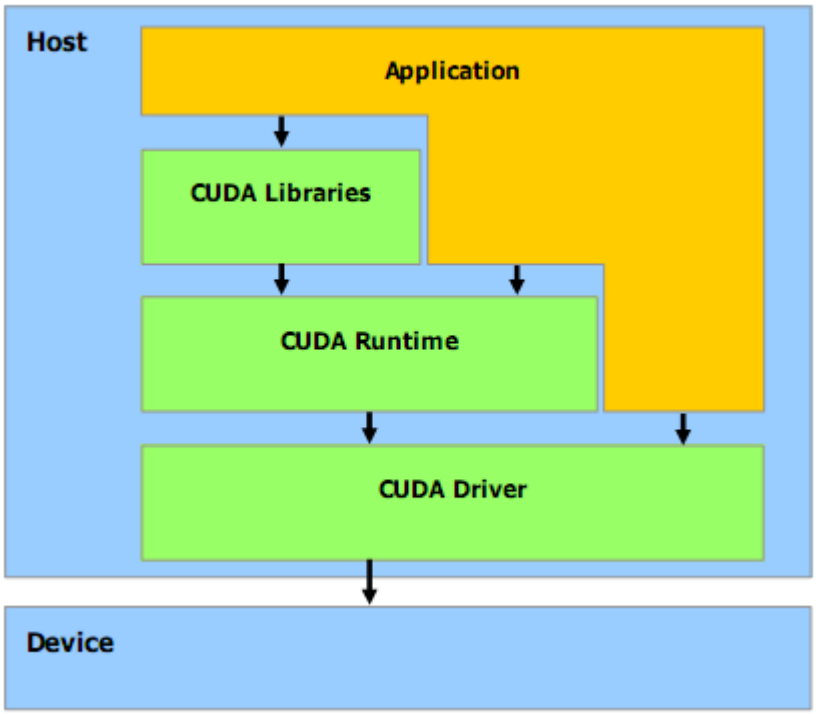
實作目標，也保留整合異質性運算平台的空間。至於是選擇 NVIDIA CUDA 或是 AMD Stream，這將取決於上層的 GPU Data Processing Library 以及 Dynamic Code Compilation Framework 在各個平台上實作的可行性及難易度。

結果與討論

a. 評估繪圖晶片運算平台（NVIDIA CUDA 或 AMD Stream）。

NVIDIA CUDA 評估

綜觀來看，如下圖所示，NVIDIA CUDA 提供了一整套的開發框架，包括語言的定義、編譯器、函式庫、執行時期、驅動程式等。圖中 Host 指的是中央處理器，或者說是整台機器上執行的部份；而 Device 則是指繪圖晶片，或者說是在顯示卡上執行的部份。



接下來我們簡單介紹一下 CUDA 語言的特性及開發及執行的流程。以下我們用一個將兩個整數陣列內容相加的程式當範例，來說明撰寫 CUDA 的程式與撰寫傳統 C/C++的程式的差異。

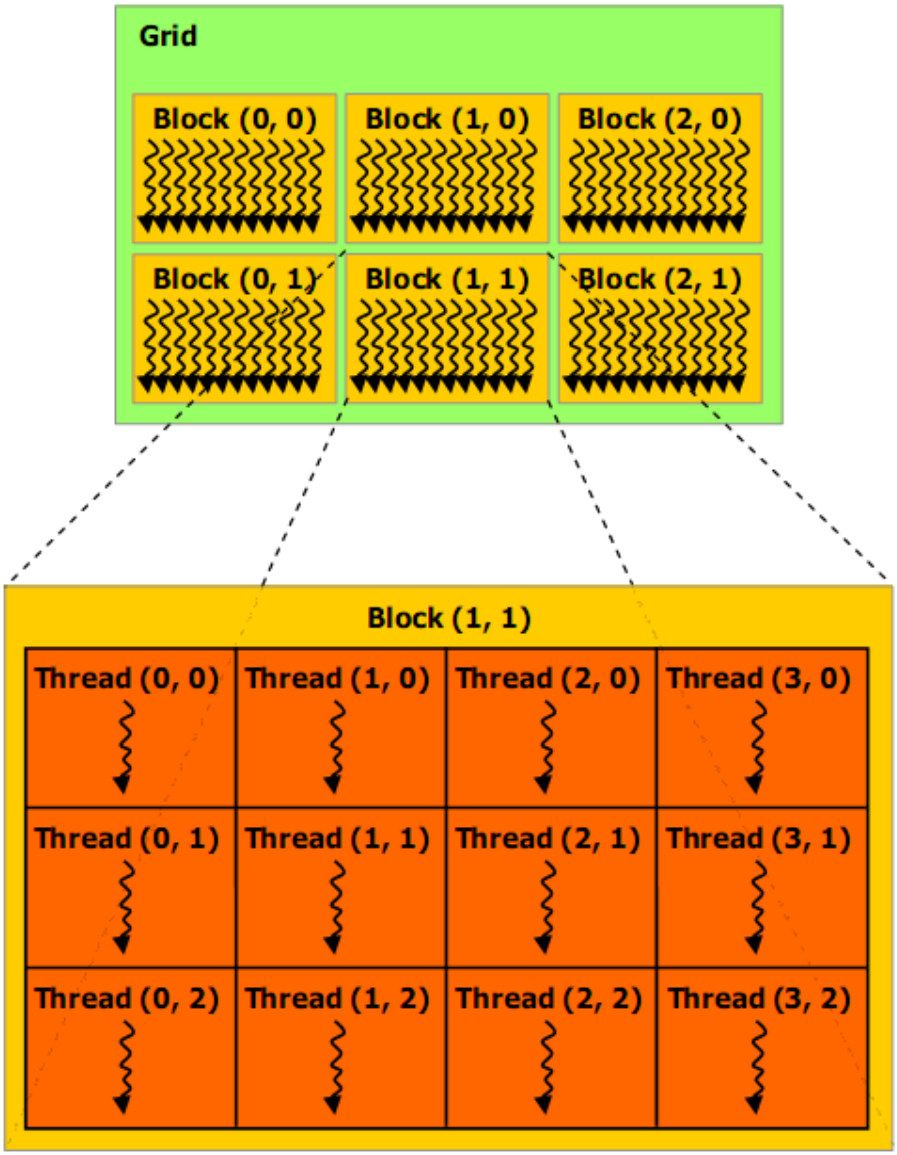
傳統 C/C++的整數陣列相加

```
int main()
{
    int A[N], B[N], C[N];
    …// initialize A,B,C
    for(int i=0;i<N;++i)
    {
        C[i] = A[i] + B[i]; // element by element
    }
}
```

CUDA 的整數陣列相加

```
__global__ void vecAdd(int* A, int* B, int* C)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x; // calculate the
global thread index
    C[i] = A[i] + B[i];
}
int main()
{
    int A[N], B[N], C[N];
    …// initialize A,B,C
    vecAdd<<<1, N>>>(A, B, C);
}
```

從上面兩個程式的範例裡，我們可以發現，CUDA 的程式基本上語法跟 C/C++是十分類似的，但需要注意的是，在 CUDA 的程式裡，我們宣告了一個函式叫 vecAdd，而且在這個函式的宣告前加了__global__這個關鍵字，意思就是告訴編譯器，這個函式是需要在繪圖晶片上執行的；然後在 main 函式裡，我們用 vecAdd<<<1, N>>>的指令來呼叫這個函式在繪圖晶片上執行。這裡的 1 及 N 分別指的「區塊」的數目，以及是每個「區塊」內「執行緒」的數目。可是什麼是「區塊」跟「區塊內的執行緒」呢？我們用下面一張圖來解釋：



在繪圖晶片上，透過 CUDA 我們可以一次產生上萬個、甚至是上百萬個執行緒；為了方便排程管理及硬體設計，這些執行緒（Thread）是有結構性的被組成一個個的「區塊（Block）」。每個區塊都有同樣數目的執行緒，許多個區塊組成一個網格（Grid）。所以在呼叫繪圖晶片上的函式時，我們需要指定區塊的數目，每個區塊內執行緒的數目，這樣繪圖晶片就會動態的配置足夠的硬體資源來執行目標函式。

AMD Stream 評估

相較於 NVIDIA CUDA，AMD Stream 提供了一個更高階的語言介面，稱為 Brook+。在 Brook+裡，程式設計師不需要考慮執行緒的數量，也不需要考慮區塊的數量，只要宣告好資料的大小，Brook+的函式庫及執行時期就會自動的產生相對應的設定。我們用同樣陣列相加的例子來展示 Brook+程式碼的架

構：

Brook+的整數陣列相加

```
kernel void vecAdd(float a<>, float b<>, out float c<>)
{
    c = a + b;
}
int main()
{
    int A[N], B[N], C[N];
    …// initialize A,B,C
    vecAdd(A, B, C);
}
```

這樣的語法相當簡單，使用者不需要去控制執行緒，由 Brook+的編譯器自動最佳化，對於使用者來說較為友善。

NVIDIA CUDA & AMD Stream 評估比較：

Brook+的語法雖然相當簡單，但是對於效能的最佳化卻不是那麼容易，因為我們沒有控制執行緒的能力，也沒有辦法完整的控制記憶體的存取，而只能依賴 Brook+的編譯器及執行時期的自動最佳化。在需要極高效能的運算能力狀況下，AMD Stream 是相對較為弱勢的。

另外，相較於 CUDA 而言，Brook+的編譯器較不成熟，問題較多，所以目前大部份的繪圖晶片上的應用都是使用 CUDA 為主。然而，Stream 除了提供了 Brook+之外，也提供了 CAL（Compute Abstract Layer），讓程式設計師可以直接撰寫繪圖晶片上的組合語言，就可以避開 Brook+的限制；然而，此一組合語言過於接近底層，隊與程式開發者來說，難以開發以及除錯。

因此在我們未來計畫的執行中，將選擇以 NVIDIA CUDA 當作目標開發平台。

b. 評估分析平行化資料處理語言（Google Sawzall，Microsoft Dryad 或 Yahoo Pig Latin)。

Google Sawzall 評估：

Google 本身意識到 MapReduce 的不足，為了要因應真實世界中更複雜的資料處理及分析，Google 在 2006 年發表了 Sawzall 這套專為資料處理分析設計的程式語言[]，利用與 MapReduce 相類似的概念，但是加上了資料模型（Data Schema）及聚合器（Aggregator）的設計，讓原本基於 MapReduce 的程式碼可以再縮短 10 到 20 倍，同時讓程式的撰寫上更為直覺。Sawzall 已經在 Google 內部被廣泛的使用，使用的頻率甚至是超越了 MapReduce。

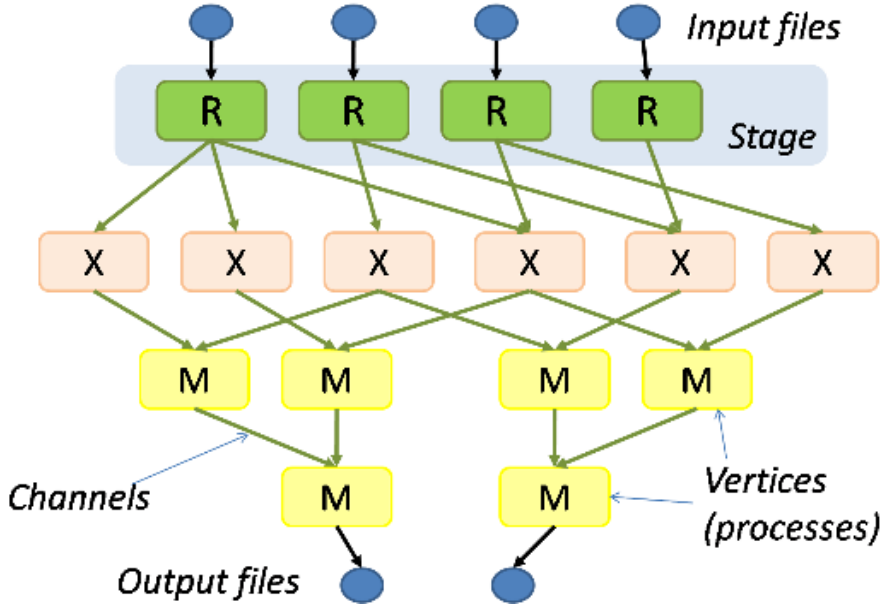
Sawzall 的程式碼相當簡潔，下圖是一個將一個陣列的的大小以及總合計算出來之程式，其實不難發現他仍然是每一個執行緒處理一筆資料的架構，所以相當適合在繪圖晶片的平行架構上實作。

```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```

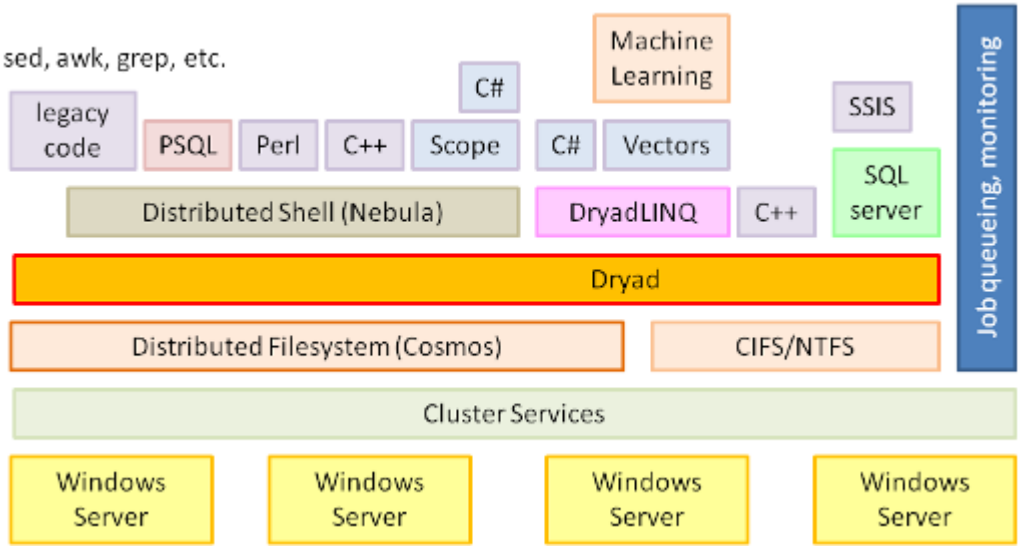
Sawzall 受了 Java 及 Pascal 的影響，在語言的設計上較為接近此兩種語言，較為高階，使用者將可以輕易的使用此一語言開發分析程式。

Microsoft Dryad 評估：

Microsoft Dryad 的程式示意圖如下：



下圖為 Dryad 的系統架構圖，其中可以看到，藉由此一個分層的架構，將可以支援多種不同程式語言。然而，Dryad 的最主要的缺點在於，只能在 Windows Sever 上執行，因此本次計畫我們將不考慮使用 Microsoft Dryad.



Yahoo PigLatin 評估：

Yahoo 在 Hadoop 的架構之上開發 PigLatin，使得程式開發者有能力借由此一語言開發能夠處理超大量資料的程式。PigLatin 開發時注重於對 SQL 的探討，究竟 SQL 缺少的是什麼，最終認為，SQL 缺少的就是 relation algebra。引此，PigLatin 基本上是以 SQL 為基礎做為延伸，因此在對於資料庫的處理方面，相較於其他解決方案，在開發上較為簡易。

以下為一個範例，當使用 Sawzall 開發時:

```
proto "querylog.proto"
static RESOLUTION: int = 5; # minutes; must be divisor of 60
log_record: QueryLogProto = input;
queries_per_degree: table sum[t: time][lat: int][lon: int] of
int;
loc: Location = locationinfo(log_record.ip);
if (def(loc)) {
  t: time = log_record.time_usec;
  m: int = minuteof(t); # within the hour
  m = m - m % RESOLUTION;
  t = trunctohour(t) + time(m * int(MINUTE));
  emit queries_per_degree[t][int(loc.lat)][int(loc.lon)] <-
1;
}
```

但是如果使用 PigLatin 開發的話，只需要以下的程式碼：

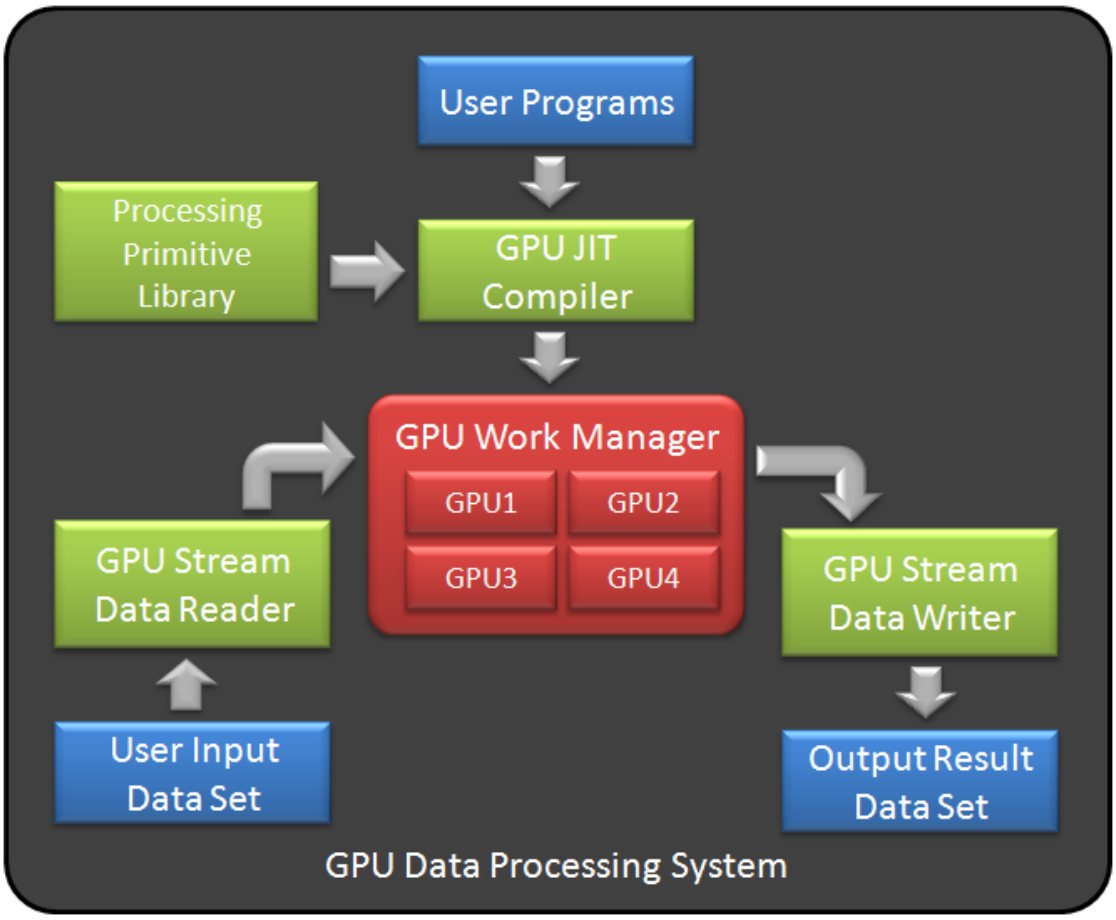
```
a = COGROUP QueryResults BY url, Pages BY url;
b = FOREACH a GENERATE FLATTEN(QueryResults.(query,
position)), FLATTEN(Pages.pagerank);
c = GROUP b BY query;
d = FILTER c BY checkTop5(*);
```

評估使用目標語言：

由於 Microsoft 的 Dryad 受限於使用的電腦作業系統，因此我們只在 Sawzall 與 PigLatin 當中做選擇。雖然 PigLatin 在 SQL 相關的應用上有非常大的優勢，然而 Sawzall 有個更好的開發出發點，希望開發的人員不需要了解如何平行的處理，只要專注於單一個資料的運算。因此這次的計畫當中，我們將採用 Sawzall 當作目標語言。

C. 完成開發 Dynamic Code Generation Framework 雛型。

我們設計的 Framework 如下圖所示:

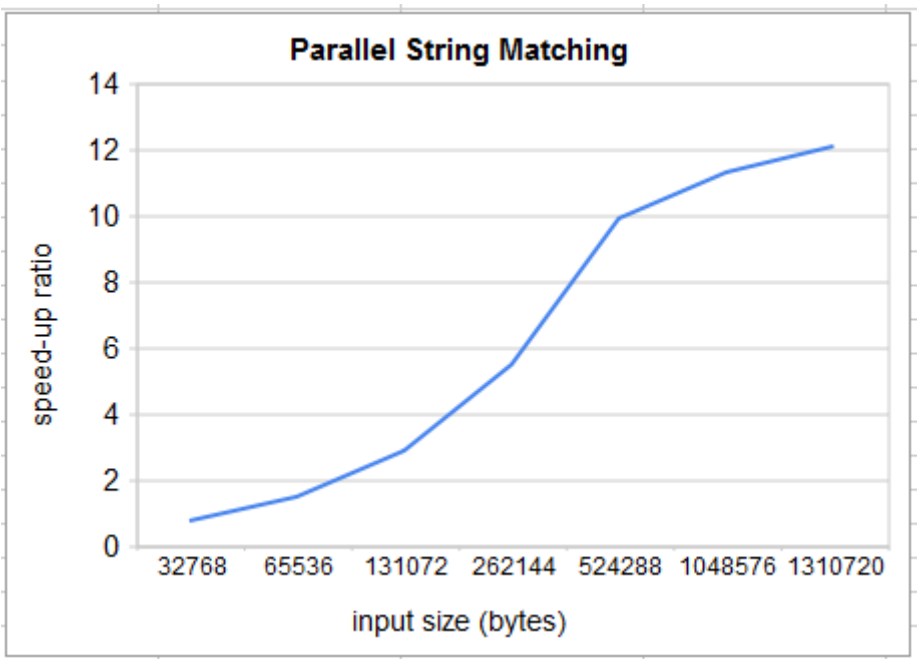


此一平台首先將使用者的程式編譯成 GPU 的程式，並且結合事先開發的 Processing Primitive Library.轉換成高效率的 GPU Work Manager。然後可以快速大量的處理使用者 input 的資料。這樣的平台設計，可以讓使用者不需要了解 GPU 運算的細節，只需要專心撰寫資料處理相關的程式，就可以藉由 Compiler 達成使用 GPU 平行運算的效能。同時藉由事先開發的 Processing Primitive Library 使用某些特殊的演算法，更加快整個運算的效能。

d. 完成開發 GPU Data Processing Library 雛型。

我們在 GPU 上面實做了 Cuckoo Hashing 的演算法。此一 hashing 演算法相對於其他 hashing 而言，最重要的就是可以在 GPU 平行處理的系統中，完美的運行。

以下為實驗的數據，將 Cuckoo Hashing 在 GPU 上實作將可以達到十倍以上的效能改進。



此一演算法可以拿來做許多的應用，結合其他使用 hashing 演算法當作基礎的演算法來使用。譬如可以將使用 hashing 當作字串比對的演算法，變成同時多字串比對；使用 hashing 當密碼驗證的演算法，可以變成同時執行多個驗證...等等。有了這樣的基礎，就可以在程式的開發上更加的有效率，使用者不再需要重新撰寫這些平行演算法，只要藉由本系統所提供的 API，即可輕易的使用這些平行演算法的效能。未來我們將持續的開發實作一些常用之 2D 或 3D 平行資料結構如 QuadTree 或 Octree，以及一些基本統計的功能。並且整合現有的 CUDPP 及現有的 CUBLAS 函式庫，以完成整個系統之開發。