(19)中華民國智慧財產局
(12)發明說明書公開本 (11)公開編號：TW 201439914 A
(43)公開日： 中華民國 103 (2014) 年 10 月 16 日

(21)申請案號：102140654 (22)申請日： 中華民國 102 (2013) 年 11 月 08 日
(51)Int. Cl.： *G06F9/45 (2006.01)* *G06F12/02 (2006.01)*
(30)優先權：2012/12/12 美國 13/712,897
(71)申請人：輝達公司 (美國) NVIDIA CORPORATION (US)
美國
(72)發明人：蘇斯尼 安卓雅娜 瑪麗亞 SUSNEA, ADRIANA MARIA (RO) ；古羅弗 文納德 GROVER, VINOD (IN) ；李 西恩 揚松 LEE, SEAN YOUNGSUNG (AU)
(74)代理人：李宗德
申請實體審查：有 申請專利範圍項數：20 項 圖式數：7 共 46 頁

(54)名稱

最佳化管理執行緒堆疊記憶體的系統、方法，及電腦程式產品

A SYSTEM, METHOD, AND COMPUTER PROGRAM PRODUCT FOR OPTIMIZING THE MANAGEMENT OF THREAD STACK MEMORY

(57)摘要

揭露用以最佳化執行緒堆疊記憶體分配的系統、方法，及電腦程式產品。方法包含以下步驟：接收一程式的原始程式碼、轉譯原始程式碼為一中間表示、分析中間表示以識別可使用在一執行緒堆疊記憶體中之第一分配記憶體空間的至少兩物件、以及藉由以對至少兩物件之第二物件的一引用取代對至少兩物件之第一物件的引用而修改中間表示。

100：方法

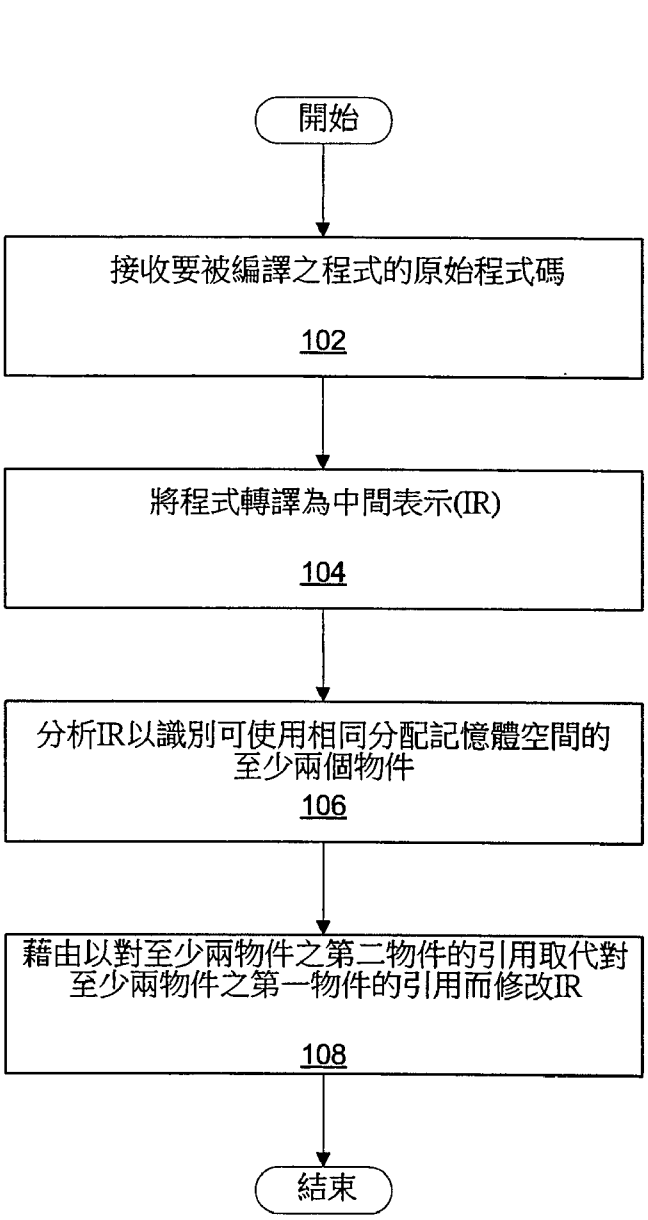
100
開始
接收要被編譯之程式的原始程式碼
102
將程式轉譯為中間表示(IR)
104
分析IR以識別可使用相同分配記憶體空間的至少兩個物件
106
藉由以對至少兩物件之第二物件的引用取代對至少兩物件之第一物件的引用而修改IR
108
結束

圖1

(19)中華民國智慧財產局
(12)發明說明書公開本 (11)公開編號：TW 201439914 A
(43)公開日： 中華民國 103 (2014) 年 10 月 16 日

(21)申請案號：102140654 (22)申請日： 中華民國 102 (2013) 年 11 月 08 日
(51)Int. Cl.： *G06F9/45 (2006.01)* *G06F12/02 (2006.01)*
(30)優先權：2012/12/12 美國 13/712,897
(71)申請人：輝達公司 (美國) NVIDIA CORPORATION (US)
美國
(72)發明人：蘇斯尼 安卓雅娜 瑪麗亞 SUSNEA, ADRIANA MARIA (RO) ；古羅弗 文納德 GROVER, VINOD (IN) ；李 西恩 揚松 LEE, SEAN YOUNGSUNG (AU)
(74)代理人：李宗德
申請實體審查：有 申請專利範圍項數：20 項 圖式數：7 共 46 頁

(54)名稱

最佳化管理執行緒堆疊記憶體的系統、方法，及電腦程式產品

A SYSTEM, METHOD, AND COMPUTER PROGRAM PRODUCT FOR OPTIMIZING THE MANAGEMENT OF THREAD STACK MEMORY

(57)摘要

揭露用以最佳化執行緒堆疊記憶體分配的系統、方法，及電腦程式產品。方法包含以下步驟：接收一程式的原始程式碼、轉譯原始程式碼為一中間表示、分析中間表示以識別可使用在一執行緒堆疊記憶體中之第一分配記憶體空間的至少兩物件、以及藉由以對至少兩物件之第二物件的一引用取代對至少兩物件之第一物件的引用而修改中間表示。

201439914

發明摘要

※ 申請案號：102140654

※ 申請日：102 年 11 月 8 日　　※IPC 分類：G06F 9/45 (2006.01)
G06F 12/02 (2006.01)

【發明名稱】

最佳化管理執行緒堆疊記憶體的系統、方法，及電腦程式產品

A SYSTEM, METHOD, AND COMPUTER PROGRAM PRODUCT FOR OPTIMIZING THE MANAGEMENT OF THREAD STACK MEMORY

【中文】

揭露用以最佳化執行緒堆疊記憶體分配的系統、方法，及電腦程式產品。方法包含以下步驟：接收一程式的原始程式碼、轉譯原始程式碼為一中間表示、分析中間表示以識別可使用在一執行緒堆疊記憶體中之第一分配記憶體空間的至少兩物件、以及藉由以對至少兩物件之第二物件的一引用取代對至少兩物件之第一物件的引用而修改中間表示。

【英文】

A system, method, and computer program product for optimizing thread stack memory allocation is disclosed. The method includes the steps of receiving source code for a program, translating the source code into an intermediate representation, analyzing the intermediate representation to identify at least two objects that could use a first allocated memory space in a thread stack memory, and modifying the intermediate representation by replacing references to a first object of the at least two objects with a reference to a second object of the at least two objects.

【代表圖】

【本案指定代表圖】：圖 1。

【本代表圖之符號簡單說明】：

100　方法

【本案若有化學式時，請揭示最能顯示發明特徵的化學式】：無。

發明專利說明書

（本說明書格式、順序，請勿任意更動）

【發明名稱】

最佳化管理執行緒堆疊記憶體的系統、方法，及電腦程式產品

A SYSTEM, METHOD, AND COMPUTER PROGRAM PRODUCT FOR OPTIMIZING THE MANAGEMENT OF THREAD STACK MEMORY

【技術領域】

【0001】 本發明係有關編譯器，特別是有關記憶體分配的最佳化。

【先前技術】

【0002】 可用於程式執行的本地記憶體為系統架構中的一個重要資源。該資源的適當管理對有效率的程式執行是相當重要的。理想上，在程式的執行過程中，只有包含針對當前指令及未來指令之資料的記憶體會被分配，而含有用於先前所執行指令之資料的任何記憶體將解除分配並回到自由記憶體池。某些記憶體由一規劃器管理，例如藉由在以C++所寫的程式中使用*malloc*及*free*指令，以明確地分配及釋放記憶體區塊。

【0003】 然而，在執行緒堆疊記憶體的情況中(亦即，分配至作用為後進先出(LIFO)佇列之程式執行緒的一部份記憶體)，規劃器一般並不管理執行緒堆疊記憶體。相反地，執行緒堆疊記憶體係隨著程式執行而動態地分配。目前的編譯器通常沒有解決執行緒堆疊記憶體之有效管理的最佳化。因此，有需要解決這個問題及/或與現有技術相關的其他問題。

【發明內容】

【0004】 揭露用以最佳化執行緒堆疊記憶體分配的系統、方法，及電腦程式產品。方法包含以下步驟：接收一程式的原始程式碼、轉譯原始程式碼為一中間表示、分析中間表示以識別可使用在一執行緒堆疊記憶體中之第一分配記憶體空間的至少兩物件、以及藉由以對至少兩物件之第二物件的一引用取代對至少兩物件之第一物件的引用而修改中間表示。

【圖式簡單說明】

【0005】 圖1描述根據一具體實施例之用以最佳化執行緒堆疊記憶體之分配的方法流程圖；

【0006】 圖2描述根據一具體實施例之用以最佳化執行緒堆疊記憶體之分配的系統；

【0007】 圖3A描述根據一具體實施例之一範例程式的原始程式碼；

【0008】 圖3B描述根據一具體實施例之對應圖3A之範例程式的中間表示；

【0009】 圖3C描述根據一具體實施例之一中間表示，其為圖3B之中間表示的修改版本；

【0010】 圖4描述根據一具體實施例之用以分析中間表示以判定哪些物件可分配至執行緒堆疊記憶體中之相同記憶體空間的方法流程圖；

【0011】 圖5描述根據一具體實施例之一平行處理單元；

【0012】 圖6描述根據一具體實施例之圖5的串流多處理器；以及

【0013】 圖7描述前述各種具體實施例之各種架構及/或功能可實施於其中的範例性系統。

【實施方式】

【0014】 某些傳統的編譯器藉由執行資料流分析並以使某些暫存器可重複用以在程式執行中的不同時間儲存不同數值的方式來指示程式指令而最佳化暫存器的分配。編譯器一般將原始程式碼(如以例如C++之高階語言所寫的程式)轉譯為中間表示(IR)，其為表示程式含意(包含執行順序)的資料結構。IR可為用於抽象機器的中間語言。IR致能編譯器執行資料流分析並在產生要由處理器執行之機器程式碼之前重新安排程式的順序。

【0015】 上述針對暫存器分配的最佳化無法以相同方式應用在執行緒堆疊中的較大記憶體結構。分配於執行緒堆疊中之物件的大小可以是可變的。相反地，暫存器具有明確定義的尺寸，例如32位元。當一數值由包含於一程式中的指令儲存於一暫存器中時，此數值需覆蓋先前分配至該暫存器的所有資料。在執行緒堆疊記憶體中的物件則有不同的表現。當一數值儲存在一物件的一元件中(如陣列的一項目)，此數值將覆蓋先前儲存在該項目中的資料。然而，物件的其他項目可仍為有效且分配至先前儲存在物件中的資料。嘗試追蹤執行緒堆疊中每一項目的資料流分析將變得非常複雜且能夠達成的最佳化將未必有效。

【0016】 以下描述的演算法係執行不同類型的資料流分析，其追蹤分配至執行緒堆疊記憶體的物件(即堆疊分配物件)是否「活」在程式的不同部份。以下的定義將用於本發明的整個揭露內容。在本說明書的前後文中，若儲存於物件中的資料有可能由當前指令或未來指令所需要，則此物件在程式的某一點(即指令)為活的。物件的*def*為儲存資料於物件中的指令。物件

之*def*的範例包含一儲存指令、具有指向物件之在指令左手邊(LHS)之一變數的一指令等。指令不需覆蓋分配至物件的全部記憶體，而僅需將資料寫入至物件的一部份。針對一物件的記憶體分配並不是物件的*def*(因為在物件中的資料並沒有被初始化)。物件的*use*為使用能夠存取堆疊分配物件之一變數的一指令。物件之*use*的範例包含一載入指令、包含指向物件之一指標的一二元操作等。

【0017】 圖1描述根據一具體實施例之用以最佳化執行緒堆疊記憶體之分配的方法100的流程圖。在步驟102，編譯器接收要被編譯的程式。程式可由高階程式語言(如C++、C#(C sharp)、Java、Cg(圖形C語言)、或某些其他高階程式語言)所寫。在步驟104，編譯器將程式轉譯為IR。IR可為中間語言，如LLVM(低階虛擬機器編譯器架構)組合語言、C、三位址碼(TAC)、或某些其他類型的中間語言。在步驟106，編譯器分析IR以識別可使用執行緒堆疊記憶體中相同分配記憶體空間的至少兩個物件。一般來說，若物件的存在時間沒有衝突，兩物件可使用相同的分配記憶體空間。換言之，只要兩物件沒有在程式中的相同點初始化或使用，兩物件可使用相同的分配記憶體空間。

【0018】 在步驟108，編譯器藉由以對至少兩物件之第二物件的引用取代對至少兩物件之第一物件的引用而修改IR。在一具體實施例中，當兩堆疊分配物件可分配至相同的記憶體空間，編譯器將以指向較大物件的新指令取代指向較小物件的指令(即根據分配至物件的記憶體尺寸)。應注意到，雖然本文提出有關最佳化執行緒堆疊記憶體之分配的各種選擇性特徵時，但這些特徵係僅用於描述目的而提出且不應以任何方式解釋為限制。

【0019】 圖2描述根據一具體實施例之用以最佳化執行緒堆疊記憶體之分配的系統200。如圖2所示，系統200包含編譯器210、程式201、中間表示(IR)202、編譯機器程式碼203。程式201、IR 202、及機器程式碼203為儲存於電腦系統之記憶體中的資料檔案。編譯器210為可執行的二元，其儲存於記憶體中且組態以由電腦系統的處理器執行。編譯器210從記憶體讀取程式201並轉譯程式201中的資料以產生IR 202。在一具體實施例中，編譯器210係實施為用於圖形處理單元之驅動器的一部份，且程式201為由應用程式所產生的著色器程式並傳送至驅動器以在圖形處理單元上執行。

【0020】 在一具體實施例中，程式201為文字檔案(如ASXII文字)，其包含具有人可讀格式的複數個指令。程式201可由高階程式語言(如C++)所寫成。IR 202包含從程式201的複數個指令提取出來成中間語言的複數個指令的表示。舉例來說，中間語言可為TAC且IR 202的每一指令表示目標處理器的一基本操作(如二元操作、邏輯操作、載入/儲存操作等)。程式201中的指令可能無法在單一時脈循環中執行。舉例來說，二元操作(如“D = A + B + C”)可能無法在單一時脈循環中由處理器執行。因此，在中間語言，上述指令可分解為中間語言中的兩個指令(如第一指令“T1 = A + B”及第二指令“D = T1 + C”)。

【0021】 一旦編譯器210已產生IR 202，編譯器對IR 202執行資料流分析。舉例來說，編譯器210可重新安排IR 202中之指令的順序。藉由改變指令的順序，編譯器能夠以更有效率的方式分配執行緒堆疊中的記憶體，對程式201中的不同變數重複使用相同的記憶體位置。編譯器可分析重新安排的指令以找出可分派給相同記憶體物件的變數，並修改IR 202中的指令使

得程式201中的不同變數可重複使用執行緒堆疊中的相同記憶體物件。一旦編譯器210已修改IR 202並對程式碼作出某些最佳化，編譯器210將編譯IR 202以產生機器程式碼203以由處理器執行。

【0022】 圖3A描述根據一具體實施例之一範例程式201的原始程式碼。如圖3A所示，程式201係以C++寫成。程式201定義常數N為2000且包含使用三個參數(即整數*n*、*m*及*k*)之函數*g*的定義。在函數*g*的每一呼叫過程中，宣告三個整數陣列(即變數*A*、*B*及*C*)以及一整數數值(即變數*result*)。變數*result*係初始化為0，而陣列*A*、*B*及*C*並沒有立即地初始化。陣列*A*首先用在程式201的行11上，其可稱作*A*的*def*。陣列*A*也用在程式201的行14，其可稱作*A*的*use*。陣列*A*在程式201的行14之後並未使用或定義。類似地，陣列*B*首先用在程式201的行18上，陣列*C*首先用在程式201的行21上，且陣列*B*及*C*用在程式201的行24上。

【0023】 函數*g*的主體包含If...Else敘述。當參數*n*小於參數*m*時，執行If...Else敘述的第一區塊(即程式201的行10-15)。當參數*n*大於或等於參數*m*時，執行If...Else敘述的第二區塊(即程式201的行17-25)。接著，變數*result*的數值由函數送回。更仔細的檢視If...Else敘述，熟此技藝者將注意到，陣列*A*在第一區塊中於第一個for迴圈內初始化(即程式201的行10-12)，且接著*A*的複數個數值在第二個for迴圈內加入至變數*result*(即程式201的行13-15)。陣列*B*在第三個for迴圈中初始化(即程式201的行17-19)，陣列*C*在第四個for迴圈中初始化(即程式201的行20-22)，且在第五個for迴圈內(即程式201的行23-25)，*B*的每一個第*k*數值係乘以*C*的每一個第*k*數值並加入至變數*result*。

【0024】 將理解到，每次函數*g*由一執行緒呼叫，函數將執行If...Else

敘述的第一區塊或第二區塊，而不是第一區塊以及第二區塊。換言之，在執行過程中，基於條件敘述("$n < m$")，陣列*A*將被定義並用於一或多指令、或是陣列*B*及*C*將被定義並用於一或多指令。有利地，編譯器可針對程式201最佳化IR 202，使得陣列*A*以及陣列*B*或陣列*C*可分配至執行緒堆疊記憶體中相同的物件。相反地，因為陣列*B*及陣列*C*可在程式中的相同點為活的(如在第五個for迴圈，來自陣列*B*及*C*兩者的數值可使用作為相同指令中的運算元)，陣列需指向執行緒堆疊記憶體中的不同物件。更一般地，若兩物件沒有在程式中的相同點為活的，則兩物件可分配至尺寸至少為兩物件中較大者之尺寸的相同記憶體空間。換言之，執行緒堆疊記憶體的一部份係分配一次且由兩物件使用。因為純量值一般由編譯器分配至暫存器，分配至執行緒堆疊記憶體的物件一般為大於單一暫存器之寬度的陣列或集合數值。

【0025】 圖3B描述根據一具體實施例之對應圖3A之範例程式201的中間表示。如圖3B所示，編譯器210已將程式201轉譯為IR 202。在行1到4，IR 202包含用於變數*A*、*B*、*C*、及*result*的分配指令。變數*A*、*B*、及*C*的物件為2000元件的整數陣列，而變數*result*的物件為單一整數。在IR 202的行5，*result*的數值係初始化為0。IR 202的行6對應程式201中主要If...Else敘述的條件敘述。因此，IR 202的行7到23對應程式201中If...Else敘述的第一區塊且行24到51對應程式201中If...Else敘述的第二區塊。IR 202的行7到14對應程式201中的第一個for迴圈、行15到23對應程式201中的第二個for迴圈、行24到31對應程式201中的第三個for迴圈、行32到39對應程式201中的第四個for迴圈、且行40到51對應程式201中的第五個for迴圈。

【0026】 圖3C描述根據一具體實施例之一IR 202-1，其為圖3B之IR

202之修改版本。如圖3C所示，圖3B之IR 202中的某些指令已修改以最佳化執行緒堆疊記憶體的分配。執行緒堆疊記憶體的部份係分配至第一陣列*A*、第二陣列*B*、及整數*result*，其係初始化為0。不同於IR 202，執行緒堆疊記憶體的部份最初並未分配至第三陣列*C*。IR 202-1中之If...Else敘述的第一區塊(即行6到22)維持與圖3B之IR 202中之If...Else敘述的第一區塊(即行7到23)相同。然而，編譯器210已修改IR 202-1中之If...Else敘述的第二區塊以最佳化執行緒堆疊記憶體的分配。將理解到，只有兩個陣列(*A*及*B*)被分配到IR 202-1。包含指向IR 202中陣列*B*之運算元的指令已由指向IR 202-1中陣列*A*的運算元所取代。類似地，包含指向IR 202中陣列*C*之運算元的指令已由指向IR 202-1中陣列*B*的運算元所取代。因此，分配在IR 202-1中的兩陣列(*A*及*B*)係分別對應至圖3B之IR 202的陣列*B*及*C*。因此，用於IR 202-1之分配執行緒堆疊記憶體的數量最多為用於IR 202-1中兩個2000元件陣列的16千位元組，相較於用於IR 202中三個2000元件陣列的24千位元組(假設變數result係分配至暫存器而非執行緒堆疊記憶體)。

【0027】 圖4描述根據一具體實施例之用以分析IR 202以判定哪些物件可分配至執行緒堆疊記憶體中之相同記憶體空間的方法400的流程圖。在步驟402，編譯器210識別IR 202中的所有堆疊分配物件。編譯器210可通過IR 202中的每一指令並判定指令是否造成執行緒堆疊中的記憶體被分配至一物件。若是，則物件係識別為堆疊分配物件。舉例來說，IR 202可包含分配指令(如"a = alloc(int, 2000)")，其分配執行緒堆疊記憶體的一部份給一物件並分派指向物件的一指標給變數*a*。

【0028】 在步驟404，針對每一堆疊分配物件，編譯器210分析IR 202

以識別能夠存取堆疊分配物件的所有變數。換言之，作為指令的結果，若在指令右側(RHS)上的一或多個變數指向物件，則表示指令左側(LHS)的變數係「指向」物件。舉例來說，若處理*a*係複製到*b*，則*b*視為指向分配至*a*的物件，因此應被追蹤以判定*a*的記憶體空間何時被使用。結果可指向堆疊分配物件的特定指令係根據IR 202所使用的中間語言而有不同。指向堆疊分配物件的指令種類包含但不限於複製指令、二元指令、轉化(如類型轉換)、PHI節點(即用以根據當前區塊的前驅而選擇一數值的特別指令)、儲存指令、及函數呼叫。在一具體實施例中，若一變數指向分配至物件之記憶體空間中的任何位置，則變數視為指向整體物件(亦即，不只是物件的一特定元件)。表格1描述用以識別能夠存取堆疊分配物件之所有變數的範例偽碼。

表格1

```
pointsTo (variable v, instruction inst) {
        if (RHS of inst includes v) {
            return LHS of inst;
        }
        else {
            return NULL;
        }
}

set accessVariables = {a}
set worklist = {a}
set alreadyChecked = null;
while worklist != null do
    v = worklist.pop()
    for instr   instructions(function) do
        if v == operand of instr then
            if altV = pointsTo (v, inst) then
                accessVariables = accessVariables   {altV}
                if altV   alreadyChecked then
                    worklist = worklist   {altV}
                end if
            end if
```

```
        end if
    end for
end while
```

【0029】 在步驟406，編譯器210對每一堆疊分配物件產生一活性網絡。堆疊分配物件的活性網絡為堆疊分配物件之*defs*(即分派一數值給物件的指令)及*uses*(即使用由物件儲存為運算元之一數值的指令)的一集合，其表示堆疊分配物件何時「活」在程式流中。將理解到，針對一物件之記憶體的分配並非該物件的*def*。若物件包含的資料有可能被當前指令或未來指令所需要，則物件在程式中的一特定點為「活的」。在一具體實施例中，針對每一堆疊分配物件，編譯器210對IR202中的指令進行迭代，並判定指令是否為物件的*def*或*use*(或兩者)。若指令為*def*或*use*，則指令加入至物件的活性網絡。表格2描述用以產生堆疊分配物件之活性網路的一範例偽碼：

表格2

```
struct LivenessWeb {
    set defs;
    set uses;
} livenessWeb

set accessVariables;   //obtained from step 404
for instr   instructions(function) do
    for v   accessVariables do
        if instr == def(v) then
            livenessWeb.addDef(v);
        end if
        if instr = use(v) then
            livenessWeb.addUse(v);
        end if
    end for
end for
```

〖0030〗 在步驟408，編譯器210追蹤每一堆疊分配物件在每一基本區塊的開始及結束是否初始化或未初始化。在一具體實施例中，編譯器210對IR 202中的每一基本區塊進行迭代並針對每一堆疊分配物件記錄何時遇到*defs*。基本區塊為具有一進入點及一離開點之程式內一部分的程式碼(即每當基本區塊中的第一指令執行時，基本區塊中剩餘的指令係依序恰好執行一次)。舉例來說，在圖3B的IR 202中，第一基本區塊從行1(即"A := alloc(in, 2000) ")延伸到行6(即"if n >= m goto L5) ")，第二基本區塊從行7(即"i := 0")延伸到行8(即"if i >= 2000 goto L2")，第三基本區塊從行9(即"t0 := i * 3")延伸到行14(即"goto L1")，以此類推。編譯器210可對每一物件使用存活性網絡(其記錄一物件之*defs*及*uses*的每一者)，以判定每一基本區塊是否包含物件的*def*。若一物件已在通往當前區塊的任何路徑上初始化，則物件係視為在當前區塊中初始化。表格3顯示用以追蹤每一堆疊分配區塊在每一基本區塊中為初始化或未初始化的一範例偽碼：

表格3

```
set allocatedObjects;   // contains all stack allocated objects
map bbToInitializedBefore; // sets of objects initialized before each block
map bbToInitializedAfter; // sets of objects initialized after each block

bool change = true
while change do
    change = false;
    for bb   basicBlocks(function) do
        workset = null;
        for predBB   predecessors(bb) do
            workset = workset   bbToInitializedAfter(predBB);
        end for
        if workset != bbToInitializedBefore.find(bb) then
            change = true;
            bbToInitializedBefore = bbToInitializedBefore   {(bb, workset)};
        end if
```

```
            for instr  instructions(bb) do
                for allocatedObject  allocatedObjects do
                    if instr  defs then
                        workset = workset  {allocatedObject};
                    end if
                end for
            end for
            if workset != bbToInitializedAfter.find(bb) then
                change = true;
                bbToInitializedAfter = bbToInitializedAfter  {(bb, workset)};
            end if
        end for
    end while
```

【0031】 在步驟410，編譯器210判定哪些堆疊分配物件在IR 202中的相同點為活的。在步驟406獲得活性網絡並在步驟408追蹤哪些物件在每一基本區塊的開始及結束為初始化及未初始化後，編譯器210維持在每一基本區塊的開始及結束為活的之一組堆疊分配物件。將理解到，步驟408僅判定物件何時初始化(即由物件的*defs*所表示)，但在步驟410中，編譯器基於物件的*uses*判定物件何時不再被需要。編譯器210藉由以相反順序對函數中的區塊及指令進行迭代(其考量到物件的*defs*及*uses*)而更新活的一組堆疊分配物件。若在IR 202中任何點有兩堆疊分配物件同時為活的，則這對物件可標記為衝突。每一堆疊分配物件係關聯於與該堆疊分配物件衝突之其他堆疊分配物件的一列表。表格4顯示用以判定哪些堆疊分配物件在相同點為活的之一範例偽碼：

表格4

```
set allocatedObjects;  // contains all stack allocated objects
map bbToBeforeSet; // sets of objects live before each block state
map bbToInitializedBefore; // sets of objects initialized before each block
map bbToInitializedAfter; // sets of objects initialized after each block
```

```
bool change = true
while change do
    change = false;
    for bb   basicBlocks(function) do
        workset = null;
        for succBB   successors(bb) do
            workset = workset   bbToBeforeSet(succBB);
        end for
        workset = workset   bbToInitializedAfter(bb);
        for allocatedObject   allocatedObjects do
            bool initialized = (allocatedObject   bbToInitializedAfter(bb));
            if initialized & allocatedObject   bbToInitializedAfter(bb) then
                initDef = first def in bb;
            end if
            for instr   instructions(bb) do // in reverse order
                if (instr   defs) & (initDef == instr) then
                    workset = workset {allocatedObject};
                end if
                if (instr   uses) & initialized then
                    workset = workset   {allocatedObject};
                    for otherObject   workset do
                        if otherObject == allocatedObject then
                            continue;
                        end if
                        allocatedObject.addConflicts(otherObject);
                        otherObject.addConflicts(allocatedObject);
                    end for
                end if
            end for
        end for
        if workset != bbToBeforeSet.find(bb) then
            change = true;
            bbToBeforeSet = bbToBeforeSet   {(bb, workset)};
        end if
    end for
end while
```

【0032】 在步驟412，編譯器210判定哪些物件可分配至相同的記憶體空間。若兩物件不衝突，則兩物件可分配至相同的記憶體空間。在步驟412，編譯器210判定實際上需要多少分配、每一分配的尺寸、及哪些物件

將使用每一分配空間。在一具體實施例中，只有具有相同類型及排列的物件被分配到相同的記憶體空間。舉例來說，若有具有排列4的兩個整數分配陣列，一陣列具有2000整數的尺寸且另一陣列具有3000整數的尺寸，且兩分配陣列並不衝突，則兩個物件可使用3000整數的單一分配空間。然而，若陣列具有不同類型(例如一個整數陣列及一個雙精度陣列)或是具有不同的排列，則物件不會被分配到相同的記憶體空間。在另一具體實施例中，具有不同類型及排列的物件可分配至相同的記憶體空間。

【0033】 在一具體實施例中，保持較大的分配且相同類型的較小分配係使用分配給較大物件的記憶體空間。針對IR 202中的每一物件類型，編譯器210以遞減的順序依照尺寸分類該類型的堆疊分配物件。維持將繼續存在於程式碼中的物件組。每一物件維持對將取代物件(若物件被取代)之堆疊分配物件的一指標、以及將使用物件的分配記憶體空間(若物件繼續存在IR 202中)的一組堆疊分配物件。較大的物件將分配至執行緒堆疊記憶體，並檢查剩餘的物件以看看其是否與第一物件或與使用該記憶體空間的其他物件衝突。若物件沒有衝突，則物件指派為使用相同的記憶體空間、或放置於一工作列表中。工作列表中最大的物件接著分配至執行緒堆疊記憶體，並檢查工作列表中的剩餘物件以看看其是否與第一物件或與使用該記憶體空間的其他物件衝突。若物件沒有衝突，則物件指派為使用相同的記憶體空間、或放置於一工作列表中，其係如同先前的工作列表等再次被檢查直到所有的物件分配到執行緒堆疊記憶體空間或指派為使用與堆疊分配物件相同的記憶體空間。表格5及6顯示用以判定哪些物件可分配至相同記憶體空間的一範例偽碼：

表格5

```
struct allocatedObjectStruct{
    set < allocatedObjectStruct > residents; // objects using this memory space
    allocatedObjectStruct *home;    // the object whose allocation this object
uses
    bool decided; // marks whether this object has been decided yet
    bool isArray; // marks whether this object is an array
    unsigned size; // the size if the object is an array
    unsigned alignment; // the alignment of the object
    Type type; // the objects type
}

set < allocatedObjectStruct > allocatedObjects;
set < allocatedObjectStruct > toAllocated = null;
for allocatedObject    allocatedObjects do
    if allocatedObject.decided then
        continue;
    end if
    choose(allocatedObject.type, allocatedObject.alignment);
end for
```

表格6

```
void choose(Type type, unsigned alignment) {
    vector < allocatedObjectStruct > rightType;
    for allocatedObject    allocatedObjects do
        if (allocatedObject.type == type) & (allocatedObject.alignment ==
                alignment) then
            rightType.insert(allocatedObject);
        end if
    end for
    if ! rightType.empty() then
        sort(rightType);
    end if

    set < allocatedObjectStruct > worklist;
    allocatedObjectStruct *current = null;
    repeat
        if current != null then
            repeat
                current = worklist.pop();
            until !current.decided
```

```
            toAllocate = toAllocate    {current};
            current.decided = true;
        end if
        for rightTypeObject    rightType do
            if rightTypeObject.decided then
                continue;
            end if
            if current == null then
                current = rightTypeObject;
                toAllocate = toAllocate    {current};
                current.decided = true;
                continue;
            end if
            if !current.isConflict(rightTypeObject) then
                current.addResident(rightTypeObject);
                rightTypeObject.addHome(current);
                rightTypeObject.decided = true;
            else
                worklist.push(rightTypeObject);
            end if
        end for
    until worklist.empty()
}
```

【0034】 將理解到，前文所提出的架構可針對各種不同的編譯器而實施。在一具體實施例中，架構可實施在平行處理器單元(PPU)的編譯器中，其產生機器程式碼以回應由在中央處理單元(CPU)上執行之一應用程式所產生的程式201。以下的描述將說明可用以實施前述架構之至少一部份的一個這樣的結構。

【0035】 圖5根據一具體實施例描述一平行處理單元(PPU)500。雖然在本文中所描述的平行處理器為PPU 500的範例，但應特別注意到，此類處理器僅為了描述目的而提出，可使用任何處理器來補充及/或替代之。在一具體實施例中，PPU 500係組態以在二或多個串流多處理器(SMs)550中同時

地執行複數個執行緒。執行緒(即執行的線程)為在一特定SM 550內執行之一組指令的實例化。每一SM 550(將在下文中結合圖6作更詳細的描述)可包含但不限於一或多個處理核心、一或多個載入/儲存單元(LSUs)、階層一(L1)快取、共享記憶體等等。

【0036】 在一具體實施例中，PPU 500包含輸入/輸入(I/O)單元505，其組態以經由系統匯流排502從中央處理單元(CPU)(圖未示)傳送及接收通訊(即指令、資料等)。I/O單元505可實施快速周邊組件互連(PCIe)介面以經由PCIe匯流排通訊。在其他具體實施例中，I/O單元505可實施其他種類的習知匯流排介面。

【0037】 PPU 500也包含主機介面單元510，其解碼指令並依照命令可能的指定將命令傳送到網格管理單元515或PPU 500的其他單元(如記憶體介面580)。主機介面單元510係組態以路由PPU 500之各個邏輯單元之間的通訊。

【0038】 在一具體實施例中，編碼為指令流的程式係由CPU寫入一緩衝器。緩衝器為記憶體(如記憶體504或系統記憶體)中的一區域，其可由CPU及PPU 500兩者存取(即讀取/寫入)。CPU將命令流寫入緩衝器，並接著傳送指向命令流之開始的一指標到PPU 500。主機介面單元510提供對一或多個命令流的指標給網格管理單元(GMU)515。GMU 515選擇一或多個命令流並組態以將所選的命令流組織為待處理網格的一集中池。待處理網格的集中池可包含尚未被選擇供執行的新網格以及已經部份執行且已暫停的網格。

【0039】 耦合於GMU 515及SMs 550之間的工作分配單元520係管理

有效網格的一集中池，其選擇及分派有效網格供由SMs 550執行。當一待處理網格有資格執行時(即沒有未解決的資料相依性)，待處理網格係由GMU 515轉移至有效網格集中池。當有效網格的執行由一相依性阻擋時，有效網格係轉移至待處理集中池。當網格的執行完成，網格將由工作分配單元520從有效網格集中池移除。除了從主機介面單元510及工作分配單元520接收網格，GMU 515也接收在網格的執行過程中由SMs 550所動態產生的網格。這些動態產生的網格係加入在待處理網格集中池中的其他待處理網格。

【0040】 在一具體實施例中，CPU執行實施一應用程式介面(API)的一驅動器核心，其致能在CPU上執行的一或多個應用程式排程操作供於PPU 500上執行。應用程式可包含使驅動器核心產生一或多個網格供執行的指令(即API呼叫)。在一具體實施例中，PPU 500實施一SIMD(單一指令多資料)架構，其中在一網格中的每一執行緒區塊(即執行包)係由在執行緒區塊中的不同執行緒同時在一不同的資料組上執行。驅動器核心定義由k個相關執行緒所組成之執行緒區塊，使得在相同執行緒區塊中的執行緒可經由共享記憶體交換資料。在一具體實施例中，一執行緒區塊包含32個相關執行緒，且一網格為執行相同資料流之一或多個執行緒區塊的一陣列，且不同的執行緒區塊可經由通用記憶體交換資料。在一具體實施例中，驅動器核心實施一編譯器，其在當產生執行緒以在PPU 500上執行時將進行對執行緒堆疊記憶體分配的最佳化。

【0041】 在一具體實施例中，PPU 500包含X個SMs 550(X)。舉例來說，PPU 100可包含15個不同的SMs 550。每一SM 550為多執行緒且組態以同時地執行來自一特定執行緒區塊的複數個執行緒(如32個執行緒)。SMs

550的每一者係經由交叉開關560(或其他類型的互連網路)連接至階層二(L2)快取565。L2快取565連接至一或多個記憶體介面580。記憶體介面580實施16、32、64、128位元的資料匯流排(或類似者)供高速資料轉移。在一具體實施例中，PPU 500包含U個記憶體介面580(U)，其中每一記憶體介面580(U)係連接至一對應的記憶體裝置504(U)。舉例來說，PPU 500可連接高達6個記憶體裝置504，例如第五代圖形雙倍資料率同步動態隨機存取記憶體(GDDR5 SDRAM)。

【0042】 在一具體實施例中，PPU 500實施一多階層記憶體架構。記憶體504係設置於晶片外耦合至PPU 500的SDRAM中。來自記憶體504的資料可被提取並儲存於L2快取565中，其係設置於晶片上且在各個SMs 550之間共享。在一具體實施例中，SMs 550的每一者也實施一L1快取。L1快取為專屬於特定SM 550的私用記憶體。L1快取的每一者係耦合至共享L2快取565。來自L2快取565的資料可被提取並儲存於每一L1快取中，供在SMs 550之功能性單元中處理。

【0043】 在一具體實施例中，PPU 500包含一圖形處理單元(GPU)，例如GPU 340。PPU 500係組態以接收指定著色器程式用以處理圖形資料的命令。圖形資料可定義為一組基元，如點、線、三角形、四邊形、三角條紋等。一般來說，基元包含指定基元的頂點數目(如在模型空間座標系統)以及關聯於基元每一頂點的屬性之資料。PPU 500可組態以處理圖形基元以產生訊框緩衝器(即針對顯示器之每一畫素的畫素資料)。驅動器核心實施圖形處理管線，例如由OpenGL API所定義的圖形處理管線。

【0044】 應用程式寫入針對一場景的模型資料(即頂點及屬性的彙

集)至記憶體。模型資料定義在顯示器上可見的每一個物件。接著，應用程式做出對驅動器核心的API呼叫，其請求呈現及顯示模型資料。驅動器核心讀取模型資料並寫入命令至緩衝器以執行一或多個操作來處理模型資料。命令可編碼不同的著色器程式，其包含一或多個頂點著色器、外殼著色器、幾何著色器、畫素著色器等。舉例來說，GMU 515可組態一或多個SMs 550以執行一頂點著色器程式，其處理由模型資料所定義的一些頂點。在一具體實施例中，GMU 515可組態不同的SMs 550以同時地執行不同的著色器程式。舉例來說，SMs 550的第一子集可組態以執行頂點著色器程式，而SMs 550的第二子集可組態以執行畫素著色器程式。SMs 550的第一子集處理頂點資料以產生處理過的頂點資料，並將處理過的頂點資料寫入至L2快取565及/或記憶體504。在處理過的頂點資料柵格化(即從三維資料轉換為在螢幕空間中的二維資料)以產生斷片資料後，SMs 550的第二子集執行一畫素著色器以產生處理過的斷片資料，其接著與其他處理過的斷片資料混合並寫入至記憶體504的訊框緩衝器中。頂點著色器程式及畫素著色器程式可同時地執行，以管線化的方式處理來自相同場景的不同資料，直到場景的所有模型已提供給訊框緩衝器。接著，訊框緩衝器的內容係傳送至顯示控制器供顯示於顯示裝置上。

【0045】 PPU 500可包含於一桌上型電腦、膝上型電腦、平板電腦、智慧型手機(如無線手持裝置)、個人數位助理(PDA)、數位相機、手持電子裝置等。在一具體實施例中，PPU 500係包含於單一半導體基板上。在另一具體實施例中，PPU 500係與一或多個其他邏輯單元(如精簡指令集電腦(RISC) CPU、記憶體管理單元(MMU)、數位到類比轉換器(DAC)等)一起包

含在一系統單晶片(SoC)中。

【0046】 在一具體實施例中，PPU 500可包含於一顯示卡上，其包含一或多記憶體裝置504(如GDDR5 SDRAM)。顯示卡可組態以與在桌上型電腦主機板(其包含例如北橋晶片組及南僑晶片組)上的PCIe插槽相接。在另一具體實施例中，PPU 500可為包含於主機板之晶片組(如北橋)中的整合圖形處理單元(iGPU)。

【0047】 將理解到，主執行緒可組態以在PPU 500的第一SM 550(0)上執行。此外，二或更多子執行緒可組態以在二或更多額外的SMs(如550(1)、550(2)等)上執行。主執行緒及子執行緒可藉由硬體視訊編碼器330存取儲存於記憶體中的動作向量資料。

【0048】 圖6根據一具體實施例描述圖5之串流多處理器550。如圖6所示，SM 550包含指令快取605、一或多個排程器單元610、暫存器檔案620、一或多個處理核心650、一或多個雙倍精度單元(DPUs) 651、一或多個特殊功能單元(SFUs)652、一或多個載入/儲存單元(LSUs)653、互聯網路680、共享記憶體/L1快取670、及一或多個紋理單元690。

【0049】 如前述，工作分配單元520係分派有效網格供在PPU 500之一或多個SMs 550上執行。排程器單元610從工作分配單元520接收網格並管理對每一有效網格之一或多個執行序區塊的指令排程。排程器單元610將執行緒排程，以供在平行執行緒的群組中執行，其中每一群組稱作一執行包。在一具體實施例中，每一執行包具有32個執行緒。排程器單元610可管理複數個不同的執行緒區塊，其在每一時脈循環過程中分配執行緒區塊到執行包供執行並接著排程來自複數個不同執行包的指令於各個功能性單元(即核

心650、DPUs 651、SFUs 652、及LSUs 653)上。

【0050】 在一具體實施例中，每一排程器單元610包含一或多個指令分派單元615。每一分派單元615係組態以傳送指令到一或多個功能性單元。在圖6所示的具體實施例中，排程器單元610包含兩個分派單元615，其在每一時脈循環過程中致能來自相同執行包的兩個不同指令被分派。在其他具體實施中，排程器單元610可包含單一分派單元615或額外的分派單元615。

【0051】 每一SM 550包含一暫存器檔案620，其對SM 550的功能性單元提供一組暫存器。在一具體實施例中，暫存器檔案620係在每一功能性單元之間劃分，使得每一功能性單元被分配暫存器檔案620的專屬部份。在另一具體實施例中，暫存器檔案620係於由SM 550所執行的不同執行包之間劃分。暫存器檔案620提供暫時儲存給連接至功能性單元之資料路徑的運算元。

【0052】 每一SM 550包含L個處理核心650。在一具體實施例中，SM 550包含大量的(如192等)不同處理核心650。每一核心650為全管線式單精度處理單元，其包含浮點算術邏輯單元及整數算術邏輯單元。在一具體實施例中，浮點算術邏輯單元執行浮點算術的IEEE 754-2008標準。每一SM 550也包含執行雙準度浮點算術的M個DPUs 651、執行特定功能(如複製矩形、畫素混合操作及類似者)的N個SFUs 652、以及執行共享記憶體/L1快取670與暫存器檔案620間之載入及儲存操作的P個LSUs 653。在一具體實施例中，SM 550包含64個DPUs 651、32個SFUs 652、及32個LSUs 653。

【0053】 每一SM 550包含互連網路680，其連接每一功能性單元到暫

存器檔案620及共享記憶體/L1快取670。在一具體實施例中，互連網路680為一交叉開關，其可組態以連接任何功能性單元到暫存器檔案620中的任何暫存器或共享記憶體/L1快取670中的記憶體位置。

【0054】 在一具體實施例中，SM 550係實施於GPU內。在此一具體實施例中，SM 550包含J個紋理單元690。紋理單元690係組態以從記憶體504載入紋理地圖(即紋素(texels)的二維陣列)並取樣紋理地圖以產生取樣紋理數值供用於著色器程式。紋理單元690執行紋理操作，例如使用mip-maps(即具有不同詳細程度的紋理地圖)的反雜訊操作。在一具體實施例中，SM 550包含16個紋理單元690。

【0055】 上述的PPU 500可組態以執行比傳統CPUs快很多的高度平行計算。平行計算在圖形處理、資料壓縮、生物統計、串流處理演算法、及類似者中具有優勢。

【0056】 圖7描述前述各種具體實施例之各種架構及/或功能可實施於其中的範例性系統700。如所示，系統700係提供為包含至少一中央處理器701，其連接至通訊匯流排702。通訊匯流排702可使用任何適合的協定來實施，如PCI(週邊組件互連)、PCI-Express、AGP(加速圖形連接埠)、超傳送標準(HyperTransport)、或其他任何匯流排或點對點通訊協定。系統700也包含主記憶體704。控制邏輯(軟體)及資料係儲存於主記憶體704中，其可採用隨機存取記憶體(RAM)的形式。

【0057】 系統700也包含輸入裝置712、圖形處理器706、及顯示器708(即傳統的CRT(陰極射線管)、LCD(液晶顯示器)、LED(發光二極體)、電漿顯示器或類似者)。使用者輸入可從輸入裝置712接收，其例如為鍵盤、滑

鼠、觸控板、麥克風及類似者。在一具體實施例中，圖形處理器706可包含複數個著色器(shader)模組、柵格化(rasterization)模組等。前述模組之每一者甚至皆可放置於單一半導體平台上，以形成圖形處理單元(GPU)。

【0058】 在本發明所說明的內容中，單一半導體平台可指稱單獨的個體半導體型積體電路或晶片。應注意，術語單一半導體平台亦可指稱有經增加之模擬晶片上操作的連接性的多晶片模組，且藉由利用傳統的中央處理單元(CPU)和匯流排實作而做出重大的改良。當然，各種模組亦可依使用者所需而分開放置或以半導體平台之各種組合放置。

【0059】 系統700也可包含次要儲存器710。次要儲存器710包括例如硬式磁碟機及/或可移除式儲存驅動，其代表軟式磁碟機、磁帶機、光碟機、數位光碟(DVD)機、記錄裝置、通用序列匯流排(USB)快閃記憶體。可移除式儲存驅動以習知的方式讀取及/或寫入可移除式儲存單元。

【0060】 電腦程式或電腦控制邏輯演算法可儲存於主記憶體704及/或次要儲存器710中。當執行這樣的電腦程式時，將致能系統700執行各種功能。記憶體704、儲存器710及/或任何其他儲存器皆為電腦可讀取媒體之可能範例。程式201、IR 202、IR 202-1、機器程式碼203、及編譯器210可儲存於主記憶體704及/或次要儲存器710中。編譯器210接著由處理器701執行以產生最佳化的機器程式碼203。

【0061】 在一具體實施例中，各種先前圖式之架構及/或功能可實施於中央處理器701、圖形處理器706、能勝任中央處理器701和圖形處理器706兩者之至少一部分能力的積體電路(未顯示)、晶片組(即設計以工作並銷售為用以執行相關功能之一單元的一積體電路群組等)、及/或與此有關的任何

其他積體電路之背景中。

【0062】 再者，各種先前圖式之架構和/或功能可實施於通用電腦系統、電路板系統、專用於娛樂用途的遊戲機系統、特定應用系統、及/或任何其他所需系統之背景中。舉例來說，系統700可具有桌上型電腦、膝上型電腦、伺服器、工作站、遊戲主機、嵌入式系統、及/或任何其他類型之邏輯的形式。此外，系統700可具有各種其他裝置之形式，其包括但不限於個人數位助理(PDA)裝置、行動電話裝置、電視等。

【0063】 此外，雖未顯示，系統700可為了通信目地而耦合至網路(如電信網路、區域網路(LAN)、無線網路、廣域網路(WAN)(如網際網路)、點對點網路、電纜網路等)。

【0064】 雖然上述已說明各種具體實施例，但應理解其僅藉由範例而非限制進行描述。因此，較佳具體實施例之廣度與範疇不應被任何上述例示性具體實施例限制，而應僅根據以下的申請專利範圍和其均等物所定義。

【符號說明】

100 方法

200 系統

201 程式

202 中間表示

202-1 中間表示

203 機器程式碼

210 編譯器

400 方法

500 平行處理單元

502 系統匯流排

504 記憶體

505 輸入/輸入單元

510 主機介面單元

515 網格管理單元

520 工作分配單元

550 串流多處理器

560 交叉開關

565 L2快取

580 記憶體介面

605 指令快取

610 排程器單元

615 分派單元

620 暫存器檔案

650 處理核心

651 雙倍精度單元

652 特殊功能單元

653 載入/儲存單元

670 共享記憶體/L1快取

680 互聯網路

690 紋理單元

700 系統

701 中央處理器

702 通訊匯流排

704 主記憶體

706 圖形處理器

708 顯示器

710 次要儲存器

712 輸入裝置

【生物材料寄存】

國內寄存資訊【請依寄存機構、日期、號碼順序註記】

國外寄存資訊【請依寄存國家、機構、日期、號碼順序註記】

【序列表】(請換頁單獨記載)

申請專利範圍

1. 一種方法，包含：

接收一程式的原始程式碼；

轉譯該原始程式碼為一中間表示(IR)；

分析該IR以識別可使用在一執行緒堆疊記憶體中之一第一分配記憶體空間的至少兩物件；以及

藉由以對該至少兩物件之一第二物件的一引用取代對該至少兩物件之一第一物件的引用而修改該IR。

2. 如申請專利範圍第 1 項所述之方法，其中該 IR 包含一 LLVM(低階虛擬機器編譯器架構)組合語言。

3. 如申請專利範圍第 1 項所述之方法，其中該原始程式碼包含由 C++、Java、C#、及 Cg 之至少一高階程式語言複數個指令所指定的複數個指令。

4. 如申請專利範圍第 1 項所述之方法，其中分析該 IR 包含：

識別在該IR中的複數個堆疊分配物件；

針對在該複數個堆疊分配物件中的每一堆疊分配物件，識別能夠存取該堆疊分配物件的所有變數；

判定哪些堆疊分配物件在該IR中的相同點為活的；以及

判定哪些堆疊分配物件可分配至該第一分配記憶體空間。

5. 如申請專利範圍第 4 項所述之方法，其中分析該 IR 更包含針對在該複數個堆疊分配物件中的每一堆疊分配物件產生一活性網絡。

6. 如申請專利範圍第 5 項所述之方法，其中該活性網絡包含該堆疊分配物件之 *defs* 及 *uses* 的一集合。

7. 如申請專利範圍第 1 項所述之方法，其中轉譯該原始程式碼為該 IR 係由實施於一圖形處理單元之一驅動器中的一編譯器所執行。

8. 如申請專利範圍第 7 項所述之方法，其中該原始程式碼包含一著色器程式。

9. 如申請專利範圍第 1 項所述之方法，更包含：

分析該IR以識別可使用在該執行緒堆疊記憶體中之一第二分配記憶體空間的至少兩個額外物件；以及

藉由以對該至少兩個額外物件之一第二物件的一引用取代對該至少兩個額外物件之一第一物件的引用而修改該IR。

10. 如申請專利範圍第 9 項所述之方法，其中分析該 IR 以識別至少兩個額外物件包含：

識別與使用該第一分配記憶體空間之另一物件衝突的一第一額外物件；以及

識別與使用該第一分配記憶體空間之另一物件衝突的至少一額外物件。

11. 一種儲存指令之非暫態電腦可讀儲存媒體，當該指令由一處理器執行時係使該處理器執行以下步驟：

接收一程式的原始程式碼；

轉譯該原始程式碼為一中間表示(IR)；

分析該IR以識別可使用在一執行緒堆疊記憶體中之一第一分配記憶體空間的至少兩物件；以及

藉由以對該至少兩物件之一第二物件的一引用取代對該至少兩物件之一第一物件的引用而修改該IR。

12. 如申請專利範圍第 11 項所述之非暫態電腦可讀儲存媒體，其中該 IR 包含一 LLVM(低階虛擬機器編譯器架構)組合語言。

13. 如申請專利範圍第 11 項所述之非暫態電腦可讀儲存媒體，其中分析該 IR 包含：

識別在該IR中的複數個堆疊分配物件；

針對在該複數個堆疊分配物件中的每一堆疊分配物件，識別能夠存取該堆疊分配物件的所有變數；

判定哪些堆疊分配物件在該IR中的相同點為活的；以及

判定哪些堆疊分配物件可分配至該第一分配記憶體空間。

14. 如申請專利範圍第 11 項所述之非暫態電腦可讀儲存媒體，其中分析該 IR 更包含針對在該複數個堆疊分配物件中的每一堆疊分配物件產生一活性網絡。

15. 如申請專利範圍第 11 項所述之非暫態電腦可讀儲存媒體，其中轉譯該原始程式碼為該 IR 係由實施於一圖形處理單元之一驅動器中的一編譯器所執行。

16. 一種系統，包含：

一記憶體，組態以儲存一程式的原始程式碼；以及

一處理單元，組態以：

接收該原始程式碼；

轉譯該原始程式碼為一中間表示(IR)；

分析該IR以識別可使用在一執行緒堆疊記憶體中之一第一分配記憶體空間的至少兩物件；以及

藉由以對該至少兩物件之一第二物件的一引用取代對該至少兩物件之一第一物件的引用而修改該IR。

17. 如申請專利範圍第 16 項所述之系統，其中分析該 IR 包含：

識別在該IR中的複數個堆疊分配物件；

針對在該複數個堆疊分配物件中的每一堆疊分配物件，識別能夠存取該堆疊分配物件的所有變數；

判定哪些堆疊分配物件在該IR中的相同點為活的；以及

判定哪些堆疊分配物件可分配至該第一分配記憶體空間。

18. 如申請專利範圍第 16 項所述之系統，其中該原始程式碼包含一著色器程式。

19. 如申請專利範圍第 16 項所述之系統，更包含耦合至該處理單元及該記憶體的一圖形處理單元。

20. 如申請專利範圍第 19 項所述之系統，更包含儲存於該記憶體中的一驅動器，其中該驅動器實現一編譯器，當該編譯器由該處理單元執行時係組態該處理單元接收該原始程式碼、轉譯該原始程式碼為該 IR、分析該 IR、及修改該 IR。

5

圖式

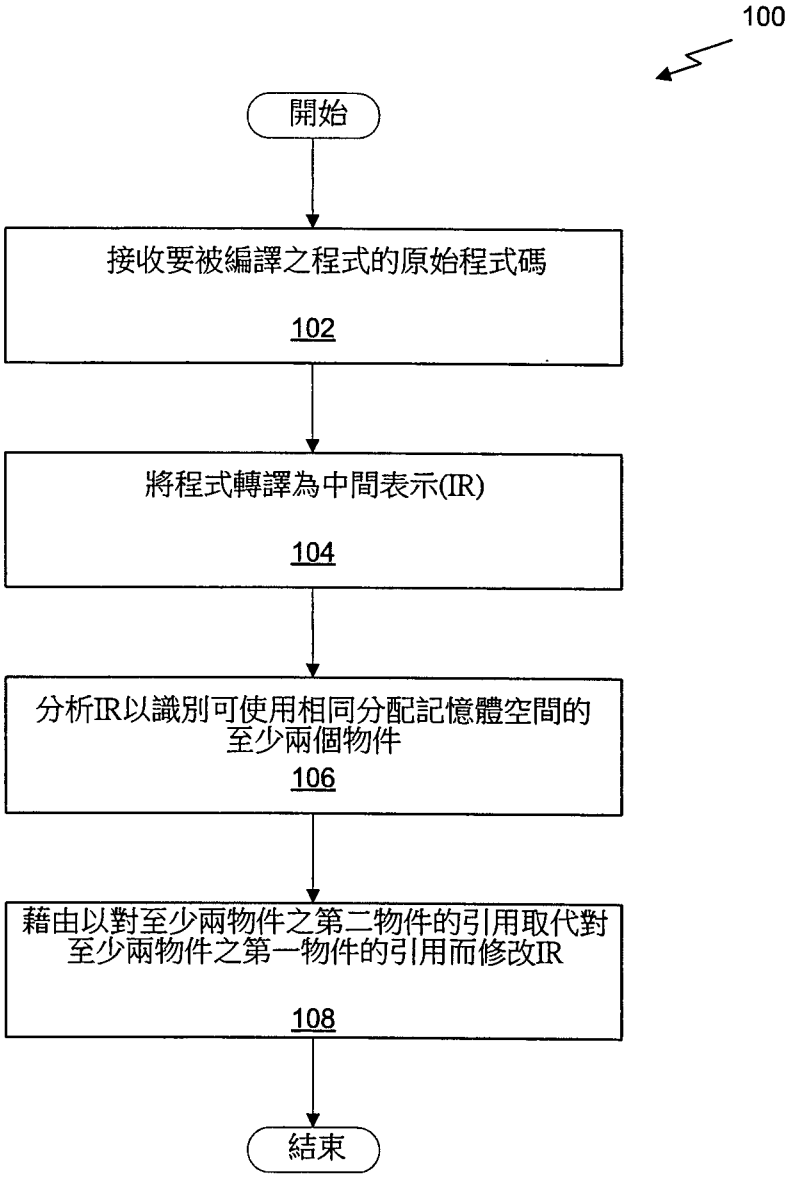
100
開始
接收要被編譯之程式的原始程式碼
102
將程式轉譯為中間表示(IR)
104
分析IR以識別可使用相同分配記憶體空間的至少兩個物件
106
藉由以對至少兩物件之第二物件的引用取代對至少兩物件之第一物件的引用而修改IR
108
結束

圖1

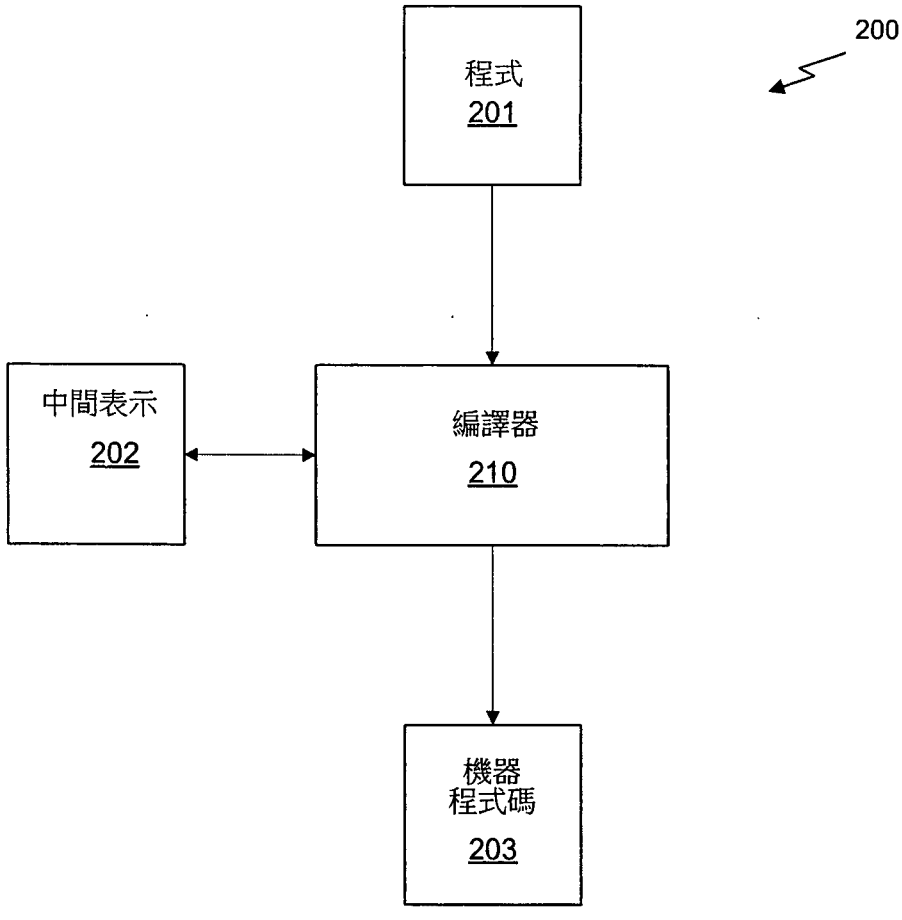

圖2

程式 201

```
1   #define N 2000
2
3   extern "C" int g(int n, int m, int k)
4   {
5
6       int A[N], B[N], C[N];
7       int result = 0;
8
9       if (n < m) {
10          for (int i = 0; i < N; i++) {
11              A[i] = i*3;
12          }
13          for (int i = 0; i < N; i++) {
14              if (i % k) result += A[i];
15          }
16      } else {
17          for (int i = 0; i < N; i++) {
18              B[i] = i*2;
19          }
20          for (int i = 0; i < N; i++) {
21              C[i] = i*i;
22          }
23          for (int i = 0; i < N; i++) {
24              if (i % k == 0) result += B[i]*C[i];
25          }
26      }
27
28      return result;
29  }
```

圖3A

中間表示 202

```
 1          A := alloc(int, 2000)
 2          B := alloc(int, 2000)
 3          C := alloc(int, 2000)
 4          result := alloc(int, 1)
 5          result := 0
 6          if n >= m goto L5
 7          i := 0
 8    L1    if i >= 2000 goto L2
 9          t0 := i * 3
10          t1 := &A
11          t2 := t1 + i
12          *t2 := t0
13          i := i + 1
14          goto L1
15    L2    i := 0
16    L3    if i >= 2000 goto L12
17          t0 := i % k
18          if t0 == 0 goto L4
19          t0 := &A
20          t1 := t0 + i
21          result := result + *t1
22    L4    i := i + 1
23          goto L3
24    L5    i := 0
25    L6    if i >= 2000 goto L7
26          t0 := i * 2
27          t1 := &B
28          t2 := t1 + i
29          *t2 := t0
30          i := i + 1
31          goto L6
32    L7    i := 0
33    L8    if i >= 2000 goto L9
34          t0 := i * i
35          t1 := &C
36          t2 := t1 + i
37          *t2 := t0
38          i := i + 1
39          goto L8
40    L9    i := 0
41    L10   if i >= 2000 goto L12
42          t0 := i % k
43          if t0 > 0 goto L11
44          t0 := &B
45          t1 := t0 + i
46          t2 := &C
47          t3 := t2 + i
48          t4 := *t1 * *t2
49          result := result + t4
50    L11   i := i + 1
51          goto L10
52    L12   return result
```

圖3B

中間表示 202-1

```
 1          A := alloc(int, 2000)
 2          B := alloc(int, 2000)
 3          result := alloc(int, 1)
 4          result := 0
 5          if n >= m goto L5
 6          i := 0
 7    L1    if i >= 2000 goto L2
 8          t0 := i * 3
 9          t1 := &A
10          t2 := t1 + i
11          *t2 := t0
12          i := i + 1
13          goto L1
14    L2    i := 0
15    L3    if i >= 2000 goto L12
16          t0 := i % k
17          if t0 == 0 goto L4
18          t0 := &A
19          t1 := t0 + i
20          result := result + *t1
21    L4    i := i + 1
22          goto L3
23    L5    i := 0
24    L6    if i >= 2000 goto L7
25          t0 := i * 2
26          t1 := &A
27          t2 := t1 + i
28          *t2 := t0
29          i := i + 1
30          goto L6
31    L7    i := 0
32    L8    if i >= 2000 goto L9
33          t0 := i * i
34          t1 := &B
35          t2 := t1 + i
36          *t2 := t0
37          i := i + 1
38          goto L8
39    L9    i := 0
40    L10   if i >= 2000 goto L12
41          t0 := i % k
42          if t0 > 0 goto L11
43          t0 := &A
44          t1 := t0 + i
45          t2 := &B
46          t3 := t2 + i
47          t4 := *t1 * *t2
48          result := result + t4
49    L11   i := i + 1
50          goto L10
51    L12   return result
```

圖3C

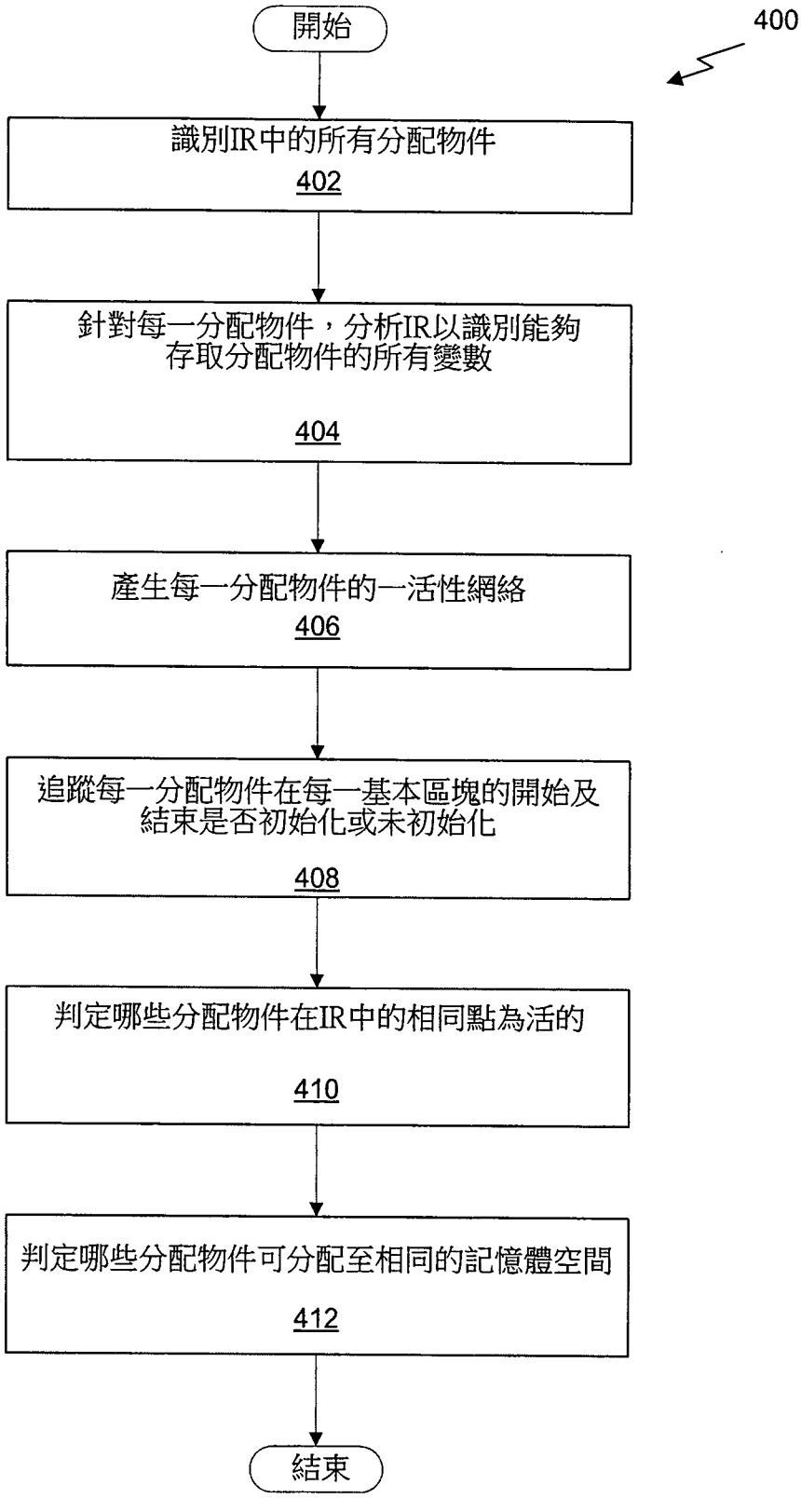
400
開始
識別IR中的所有分配物件
402
針對每一分配物件，分析IR以識別能夠存取分配物件的所有變數
404
產生每一分配物件的一活性網絡
406
追蹤每一分配物件在每一基本區塊的開始及結束是否初始化或未初始化
408
判定哪些分配物件在IR中的相同點為活的
410
判定哪些分配物件可分配至相同的記憶體空間
412
結束

圖4

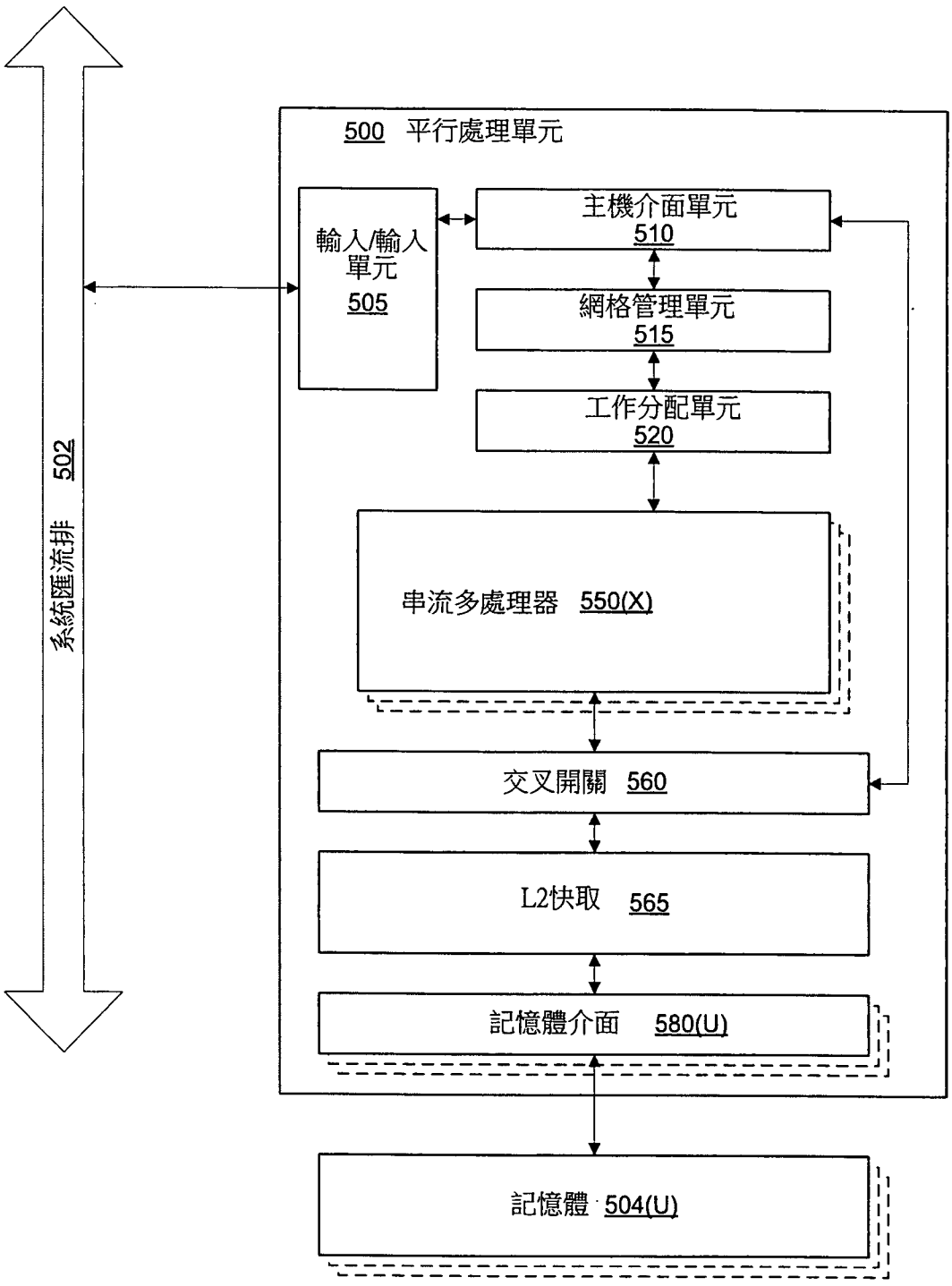
500 平行處理單元
輸入/輸入單元 505
主機介面單元 510
網格管理單元 515
工作分配單元 520
串流多處理器 550(X)
交叉開關 560
L2快取 565
記憶體介面 580(U)
記憶體 504(U)
系統匯流排 502

圖5

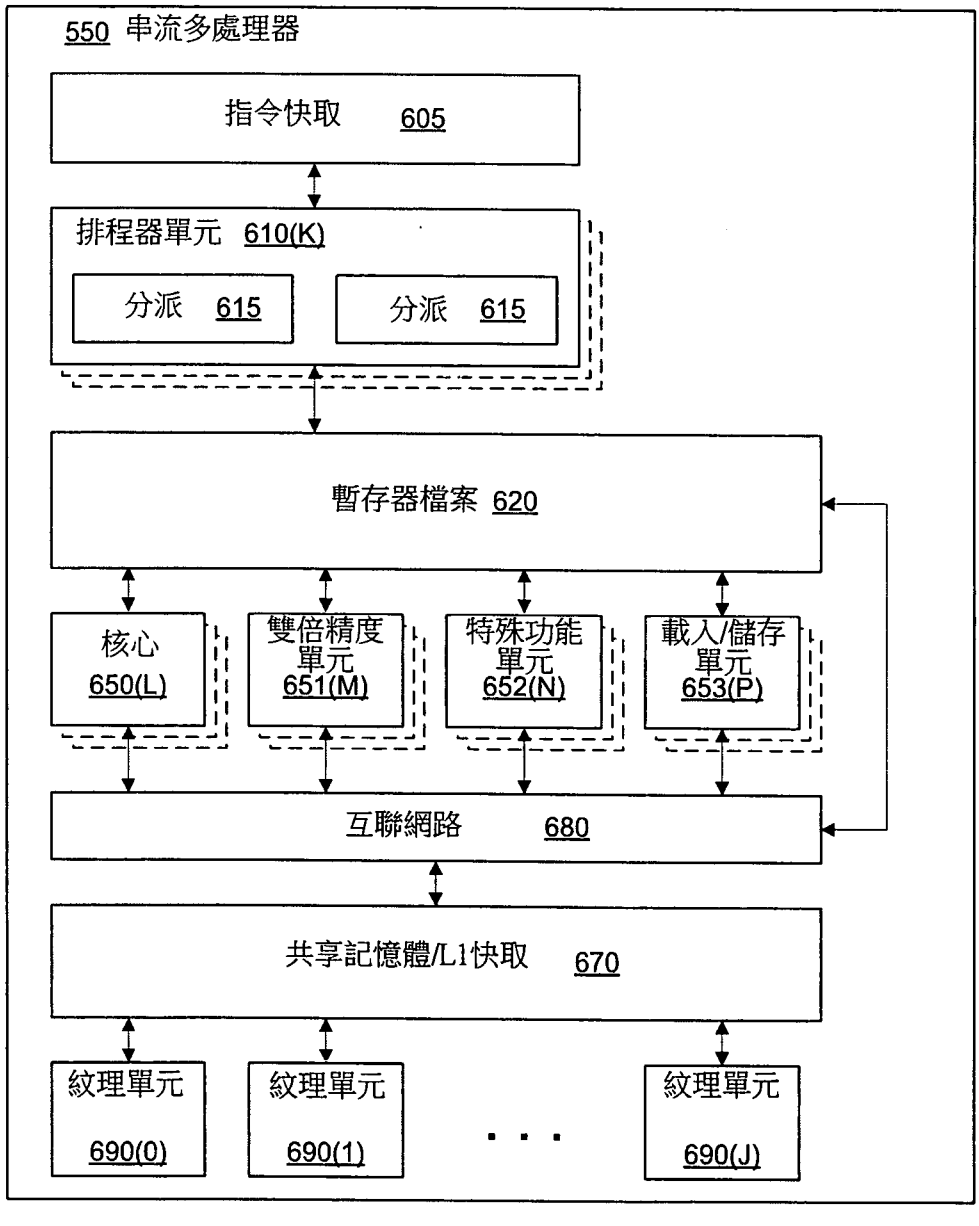
550 串流多處理器
指令快取 605
排程器單元 610(K)
分派 615
分派 615
暫存器檔案 620
核心 650(L)
雙倍精度單元 651(M)
特殊功能單元 652(N)
載入/儲存單元 653(P)
互聯網路 680
共享記憶體/L1快取 670
紋理單元 690(0)
紋理單元 690(1)
紋理單元 690(J)

圖6

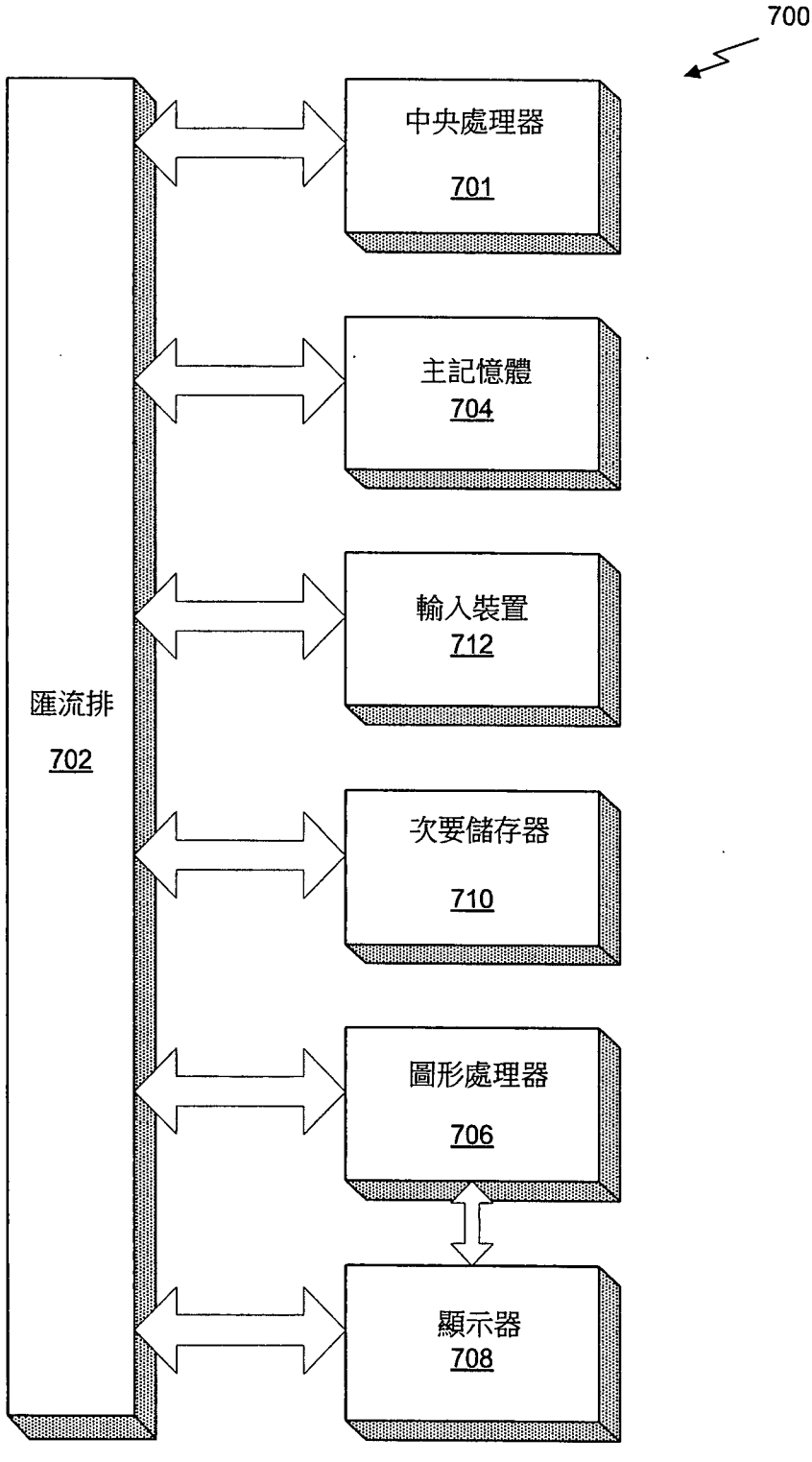
700
中央處理器
701
主記憶體
704
輸入裝置
712
匯流排
702
次要儲存器
710
圖形處理器
706
顯示器
708

圖7