I470425

(19)中華民國智慧財產局

(12)發明說明書公告本 **(11)證書號數：TW I470425 B**

(45)公告日： 中華民國 104 (2015) 年 01 月 21 日

(21)申請案號：100145350 (22)申請日： 中華民國 100 (2011) 年 12 月 08 日

(51)Int. Cl.： *G06F11/36 (2006.01)* *G06F9/38 (2006.01)*
G06F9/50 (2006.01)

(30)優先權：2010/12/25 美國 12/978,557

(71)申請人：英特爾股份有限公司 (美國) INTEL CORPORATION (US)
美國

(72)發明人：賽吉 大衛 SAGER, DAVID J. (US)；沙珊卡 路奇拉 SASANKA, RUCHIRA (LK)；賈柏 朗 GABOR, RON (IL)；瑞金 修莫 RAIKIN, SHLOMO (IL)；紐茲曼 約瑟夫 NUZMAN, JOSEPH (IL)；皮雷 李歐 PELED, LEEOR (IL)；多摩傑森 DOMER, JASON A. (US)；金何碩 KIM, HO-SEOP (KR)；吳玉芬 WU, YOUFENG (US)；山田耕一 YAMADA, KOICHI (JP)；奈 廷復 NGAI, TIN-FOOK (HK)；陳 霍華 CHEN, HOWARD H. (US)；伯霸 加亞倫 BOBBA, JAYARAM (IN)；庫克 傑佛瑞 COOK, JEFFREY J. (US)；賽克 奧斯瑪 SHAIKH, OSMAR M. (US)；史瑞尼維斯 蘇瑞 SRINIVAS, SURESH (US)

(74)代理人：林志剛

(56)參考文獻：

US 6711667B1 US 20100205599A1

審查人員：何偉權

申請專利範圍項數：18 項 圖式數：15 共 135 頁

(54)名稱

硬體及軟體系統用之將程式自動分解為多重平行緒之系統，裝置及方法

SYSTEMS, APPARATUSES, AND METHODS FOR A HARDWARE AND SOFTWARE SYSTEM TO AUTOMATICALLY DECOMPOSE A PROGRAM TO MULTIPLE PARALLEL THREADS

(57)摘要

說明硬體及軟體系統用之將程式自動分解為多重平行緒之系統、裝置、及方法。在一些實施例中，系統及裝置執行原始碼分解及/或產生緒執行的方法。

Systems, apparatuses, and methods for a hardware and software system to automatically decompose a program into multiple parallel threads are described. In some embodiments, the systems and apparatuses execute a method of original code decomposition and/or generated thread execution.

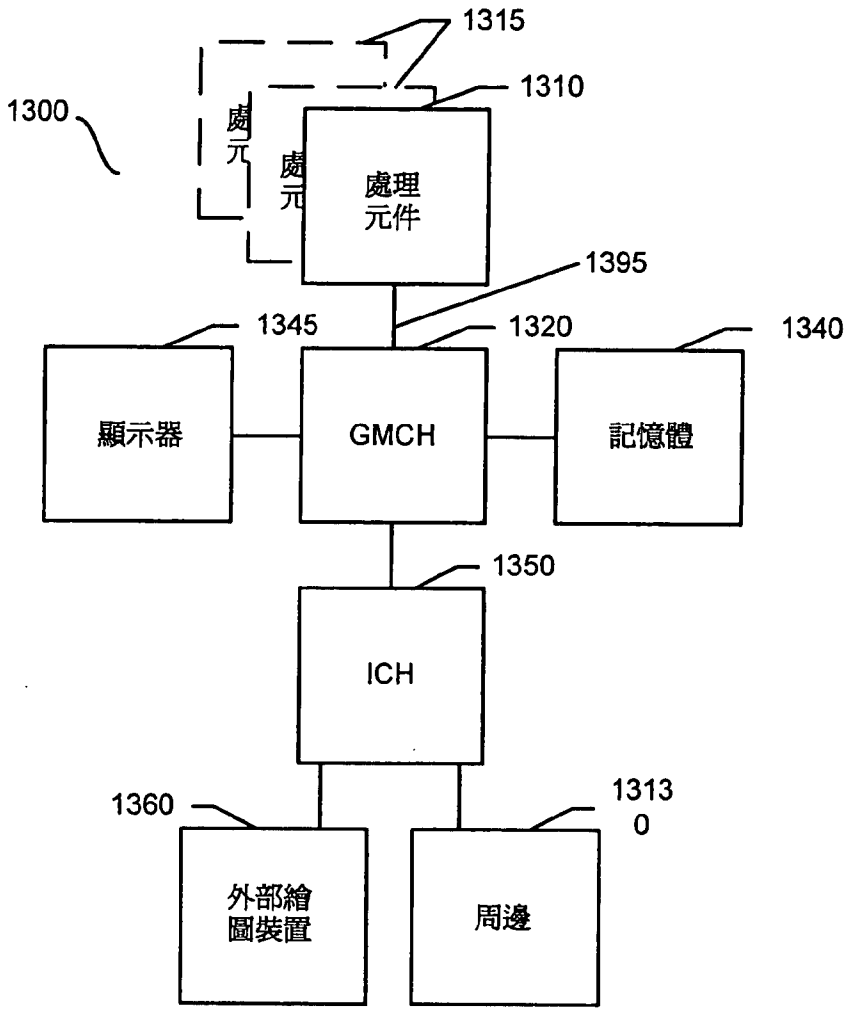

圖13

1300・・・系統

1310・・・處理元件

1315・・・處理元件

1320・・・繪圖記憶體控制器中樞(GMCH)

1340・・・記憶體

1345・・・顯示器

1350・・・輸入/輸出控制器中樞

1360・・・外部繪圖裝置

1395・・・前端匯流排

公告本

發明專利說明書

（本申請書格式、順序，請勿任意更動，※記號部分請勿填寫）

※申請案號：100145350

※申請日：100 年 12 月 08 日　　※IPC 分類：G06F 11/36 (2006.01)
G06F 9/38 (2006.01)
G06F 9/50 (2006.01)

一、發明名稱：（中文／英文）

硬體及軟體系統用之將程式自動分解爲多重平行緒之系統，裝置及方法

Systems, apparatuses, and methods for a hardware and software system to automatically decompose a program to multiple parallel threads

二、中文發明摘要：

說明硬體及軟體系統用之將程式自動分解爲多重平行緒之系統、裝置、及方法。在一些實施例中，系統及裝置執行原始碼分解及/或產生緒執行的方法。

三、英文發明摘要：

Systems, apparatuses, and methods for a hardware and software system to automatically decompose a program into multiple parallel threads are described. In some embodiments, the systems and apparatuses execute a method of original code decomposition and/or generated thread execution.

四、指定代表圖：

（一）本案指定代表圖為：第（13）圖。

（二）本代表圖之元件符號簡單說明：

1300：系統

1310：處理元件

1315：處理元件

1320：繪圖記憶體控制器中樞（GMCH）

1340：記憶體

1345：顯示器

1350：輸入/輸出控制器中樞

1360：外部繪圖裝置

1395：前端匯流排

五、本案若有化學式時，請揭示最能顯示發明特徵的化學式：無

六、發明說明：

〔優先權主張〕

這份部分延續申請案主張序號 12/646,815 號、名稱爲「Systems, Methods, and Apparatuses for Parallel Computing（平行運算的系統、方法、及裝置）」的正式申請案之優先權日，其本身主張優先權於序號 12/624,804 號、2009 年 11 月 24 日提出申請、名稱爲「System, Methods, and Apparatuses To Decompose A Sequential Program Into Multiple Threads, Execute Said Threads, and Reconstruct The Sequential Execution（將序列程序分解爲多重執行緒、執行所述執行緒、及重建該序列執行之系統、方法、及裝置）」的正式申請案，該案主張優先權於序號 61/200,103 號、2008 年 11 月 24 日提出申請、名稱爲「Method and Apparatus To Reconstruct Sequential Execution From A Decomposed Instruction Stream（自分解指令流重建序列執行之方法及裝置）」的臨時申請案。

【發明所屬之技術領域】

本發明之實施例一般係相關於資訊處理之領域，特別是相關於運算系統與微處理器中的多重緒執行之領域。

【先前技術】

單執行緒處理器於近十年藉由開發指令階層平行性（instruction level parallelism, ILP）而顯現出顯著的性能

改善。然而，這種平行性有時難以開發且需要複雜硬體架構，導致過高的功率消耗與設計複雜度。此外，複雜度與功率的增加將減低回報。晶片多處理器（CMP）作爲在合理的功率預算下提供進一步的處理器性能改善的有前途選擇而嶄露鋒芒。

【發明內容及實施方式】

在隨後的說明中，許多具體細節將予以闡明。然而，當知本發明之實施例可不須這些具體細節而實踐。在其它實例中，爲了不混淆對實施方式的理解，不對眾知電路、結構及技術多加陳述。

於此說明書中提及「一個實施例」、「一實施例」、「一範例實施例」等詞，代表所述實施例可包括一特有特色、結構、或特徵，但每個實施例可未必包括此特有特色、結構、或特徵。此外，這類用詞未必涉及相同實施例。再者，當說明一特有特色、結構、或特徵關聯於一實施例時，這表示無論是否明確說明，將這樣的特色、結構、或特徵變更關聯於其它實施例屬於該領域熟習此項技術者之通常知識。

提供動態緒切換執行之系統、裝置、及方法詳述於後。支援此者之系統的實施例包含由一硬體包整程式（wrapper）及軟體（動態緒切換軟體）所圍繞的處理器核心。在一正常執行模式中，處理器以似乎硬體包整程式與動態緒切換軟體爲不存在而運作。例如，正常的 x86 執行

之發生。在動態緒切換執行（dynamic thread switch execution, DTSE）模式中，硬體包整程式及軟體共同工作而執行包括產生及調整流（flow）等的動態緒切換執行（其將詳述於後）。然而，首先提供高階總覽。接著將諸如「流（flows）」及「熱」碼的這些主題的一部分予以更詳細解釋規範。

下述細節說明一硬體包整程式連同軟體之結合，硬體包整程式在整體提交之前對自正確的儲存處及特定的其它物件取得資料之每個載入予以查對，該軟體使用最近觀察的碼行爲而對於難以僅就碼本身判斷碼之相依性予以解析，於是硬體將會查對及保證正確的執行。

下述細節說明在原始碼中許多靜態實體（instance）之識別，以致在原始指令之特定實體之間以及在非介於原始指令自身之間的相依性被瞭解。

所述軟體使用一完整但可能並非絕對正確之在一被界定之靜態碼子集（稱爲流（flow））中之相依性之理解而將靜態碼分離爲在該流中的分離的「軌跡（track）」，其於軌跡之間沒有相依性（或只有特定允許之相依性）。

此外，說明包括一個或多個如後述的維護操作：在因非預期控制流或非正確載入結果或其它查對失敗而退出前，採取變得較期望動態執行爲短的流之額外描述檔（profile）；刪除在此方面突然變壞的流；繼續尋找對於效能爲重要的新的或取代的流；以及當有新的描述檔資訊時將碼重新分離成軌跡，並將觀察到的不比原始碼執行好的

流予以刪除。

DTSE 硬體將形成一順序清單並且將之寫入中階快取記憶體。DTSE 硬體包括具有關聯於此軌跡元資料清單的實體位址之暫存器。我們可能必須考慮壓縮載入與儲存位址以保持頻寬低下。

我們需要能夠追蹤正執行一載入或儲存的邏輯處理器，以進行載入執行以及上級儲存處理。

DTSE 邏輯亦取得在引退（retirement）時的分支方向位元以及採用間接分支的目標（包括返回（return））。這用於快閃概析（Flash Profiling）以及用於支援暫存器狀態恢復的資料。

I. 高階概述

如上所述，在此詳述的處理器可運作於各核心個別自身運作的一傳統模式。當其中一個核心爲停止「（off）」或停機時，處理器核心也可用於在 DTSE 模式中概析、緒化、以及運行（running）緒碼。即便作業系統認定該核心處於睡眠模式，停止（halt）流將核心之所有權自 OS 切換至 DTSE 模式。

圖 1 爲圖表展示 DTSE 作業之一範例實施例。此由在圖式中成爲次要核心的一閒置核心取得控制。在一些實施例中，閒置核心之控制發生於核心接收一 HALT/WMAIT 指令時。DTSE 軟體監看待進入睡眠的核心之所有緒並且取得此核心之控制。

次要核心藉偵測「熱碼（hot-code）」開始，熱碼爲指出運行於對應大量動態執行（例，用於 100K 連續指令運行的核心（kernel））的主要核心的軟體的入口點。熱碼偵測典型地於簡單硬體結構中完成。一旦偵測到熱入口點，則提供用於概析。一旦命中（藉由主要核心），此熱碼進入概析模式，意指對於一些設定的指令量（諸如 50,000 個指令）之所有載入、儲存、及分支予以概析。一旦概析結束，則進行緒的產生。緒產生分析熱碼並且基於概析資訊而產生用於緒執行模式的二個緒。一旦完成，則將入口點（hot IP）提供作爲一 DTSE 模式入口點。在下次原始碼（運行於主要核心）命中此入口點時，則切換成雙核心協作執行此碼的 DTSE 模式。若流退出此熱碼或是若偵測到違反（violation）或外部事件（例，中斷（interrupt）），則 DTSE 模式終止。該違反包括例如當雙核心於同時期儲存相異資料至相同記憶體位址。當爲違反時，緒模式退至最近的提交點並且回移至原始碼。

圖 2 繪示依據一些實施例的 DTSE 作業之一範例方法。在此範例中，利用二個核心處理緒碼。在 201，主要核心（核心 0）執行原始訊號緒碼。

在 203，另一核心（核心 1）成爲並且用做次要核心。一核心成爲次要核心（一工作緒）的方式有許多種。例如，透過硬體動態規劃的使用，諸如抓取被 OS（例，變成 C 狀態）、軟體指派、或緒（藉由 OS/驅動程式或應用程式本身）置入睡眠的核心，而能夠將次要核心使用作爲

核心之靜態分割下的次要核心。

在 205，當主要核心正執行此原始碼時，次要核心將被置入偵測階段，其等待一熱碼偵測（藉由硬體或軟體）或是一熱區域。在一些實施例中，熱碼偵測爲一硬體表，其偵測頻繁存取的熱區域，並且提供其入口 IP（指令指標）。一旦偵測到這類的熱區域入口 IP，則提供主要核心以致在下次請求此 IP 時將會觸發概析、以及將會切換執行於原始碼之緒版本，在 207。此概析擷取諸如載入地址、儲存地址、及分支的資訊，用於一預定長度的執行（例，50,000 動態指令）。

一旦概析已結束，次要核心開始緒產生階段（thread-gen）209。此階段中，次要核心產生概析區域之緒版本，在使用此概析資訊作爲引導時。緒產生提供此原始碼之緒版本，並伴隨可能的入口點。在 211，當入口點之一（標記爲「Hot IP」）爲命中（hit）時，主要核心與次要核心群再指向執行此碼之緒版本並且執行切換成不同的執行模式（有時稱爲「緒執行模式」）。在此模式中，當使用包整程式硬體以緩衝儲存記憶體載入及儲存、並對其可能的違反予以查對、以及基元性提交此狀態以在維持記憶體排序的同時提供前移時，二個緒完全分離地運作。

此執行模式可採二種方式之一而終止。可當碼退出熱區域作爲淨退出（clean exit）（執行無問題）或當發生違反而作爲髒退出（dirty exit）時終止。在 213，爲決定作出何種退出的判斷。範例的髒退出爲儲存/儲存以及載入/

儲存違反或是在不相關於第二執行模式中的一例外情境（例，浮點除以零例外、非快取記憶型儲存等）。在第二執行模式退出的時候，主要核心回至原始碼，同時次要核心回至偵測模式並等待另一熱 IP 被偵測或是一已產生區域的熱 IP 被命中。在淨退出（熱區域退出）時，原始碼從退出點接續。在髒退出（例，違反或例外）的時候，在215，主要核心回至最近的檢查點以及繼續對該處的執行。在淨與髒退出皆發生的時候，暫存器狀態自雙核心合併並且移入原始核心。

範例 DTSE 架構

圖 3 繪示一動態緒切換執行架構之一實施例。此系統之軟體與硬體方面詳細討論於後。如上所述，各核心 301 可原生支援同步多重緒（SMT）。這意指二個以上邏輯處理器可共享核心之硬體。各邏輯處理器獨自處理一編碼流，但會隨機混合來自這些編碼流的指令，以在相同硬體上執行。頻繁地，來自相異邏輯處理器的指令在核心之超純量硬體上同步執行。SMT 核心之性能以及在相同核心的邏輯處理器之數量增加。有因於此，一些重要的工作負荷將因邏輯處理器之數量增加而處理的更快。其它工作負荷因僅有邏輯處理器之數量增加而可能不會處理的更快。

有些時候系統中沒有足夠的軟體緒以取得所有邏輯處理器之優勢。此系統自動地分解一些或全部的可用軟體緒，各成爲將被同時執行的多重緒（自單一緒切換至多重緒

的動態緒），如此而取得多重（或許爲許多）的邏輯處理器的優勢。對於只是因爲增加了邏輯處理器之數目而不爲處理得更快的工作負荷將當其多數緒已分解成更大量的緒而使用更多邏輯處理器而變得速度更快。

除了「傳統」核心之外，硬體包括動態緒切換邏輯，其包括維持全域記憶體一致性 303、全域引退 307、全域暫存器狀態 309、及擷取軟體用資訊 305。此邏輯可包括在核心群本身中或是分別提供。此邏輯可執行五項功能。其一爲擷取關於被稱爲概析的運行碼的特定資訊。其二，是當原始碼運行時，硬體必須查看命中軟體已經界定之熱 IP 串流位址之執行。當其發生時，硬體迫使核心跳越至軟體已經界定之不同位址。這即是碼之緒版本被執行的方式。其三爲硬體必須與軟體一同工作以不時地有效保留原始碼串流之正確暫存器狀態作爲核心資料快取內部的全域提交點。若原始碼串流藉由軟體分解成多重緒，則可能沒有曾具有原始程式之整個正確暫存器狀態的邏輯處理器。伴隨著各全域提交點的正確記憶體狀態也應該被知道。在必要時，偕同軟體工作的硬體必須能夠恢復暫存器與記憶體二者的架構程式狀態至最近的全域提交點，如後述將討論般。其四，雖然軟體在生產正確執行的碼方面將做得相當好，有一些事件軟體無法在 100%的時間都是正確的。一適合範例便是軟體在產生碼之緒版本時，無法完美預期記憶體位址。因而緒碼將會偶爾於載入得到錯誤結果。硬體必須查對所有可能不正確的項目。若某項不正確，硬體必

須偕同軟體取得不變的程式狀態。這通常藉由恢復核心狀態至最近的全域提交狀態而完成。最後，若原始碼串流分解爲多重緒，則於原始碼中載明的記憶體的儲存將會分散在多重邏輯處理器之中，並且隨機排序地在這些邏輯處理器之間執行。動態緒切換邏輯必須保證任何其它碼串流將不能夠「查看」到如原始碼所界定的不正確的記憶體中的狀態一事被正確地執行。

在這些範例架構中，一處理器核心（「核心」）通常具有二個 SMT 緒。一般而言，在 DTSE 模式中每個核心有許多個多於二個的「邏輯處理器」。這些邏輯處理器被界定爲二種類型之一：「主要」及「工作者」。主要邏輯處理器爲軟體（即，作業系統及虛擬機器監控程式）所知。軟體將在核心的主要邏輯處理器組視爲它們實際所在的核心的 SMT 緒。主要邏輯處理器執行大部分的事，若否，則由 SMT 緒處理，特別是取得中斷。

工作者邏輯處理器並非外部軟體一般可見。它們由「保留記憶體」中的 DTSE 軟體所使用及管理。此記憶體爲一實體可定址保留記憶體系統。此保留記憶體可爲現存系統記憶體之一專屬部分或是分散的實體記憶體。此保留記憶體用於儲存 DTSE 碼並且扮演其工作記憶體。DTSE 模式中產生的資料結構儲存於此保留記憶體中。此「熱」碼也儲存於這個保留記憶體中。軌跡資料、軌跡資料目錄、以及軌跡元資料儲存於保留記憶體中。這些東西全部被實體定址。

工作者邏輯處理器不具有主要邏輯處理器所具有的全文（context），並且，因此，受限制於全處理器功能之子集。工作者邏輯處理器僅執行保留記憶體中的流中的碼並且不執行可見記憶體中的原始碼。

主要邏輯處理器可執行可見記憶體中的原始碼或隱藏記憶體中的流中的碼。在主要（邏輯處理器）所將執行的隱藏記憶體中的流中的碼係爲單軌跡碼，自 DTSE 觀點來看。

在操作上，作業系統運行主要邏輯處理器上的緒，如其目前所進行。主要邏輯處理器全都相同並且群組成「核心」。此處有許多主要邏輯處理器且各個皆爲非常高性能。藉由作業系統可用的主要邏輯處理器之數量加乘主要邏輯處理器的性能將遠勝晶片之總處理能力。

其可藉由使用所提供的主要邏輯處理器的部分（fraction）而使用晶片的整個能力（capability）。另一方面，藉由使用多數量（最好爲全部）的所提供的主要邏輯處理器，良好的編碼並有大量平行（運算）程式可更有效地使用此晶片的能力，並因此達到更高性能。

若正在運行的主要邏輯處理器並未使用所有晶片所提供的資源，那麼 DTSE 軟體將會做自動的緒分解。將會在工作者邏輯處理器上運行平行運算流。這包括，特別是，將重要的 SSE 及 AVX 碼予以分解以平行地運行於流量（through put）引擎上。這包括在那些特有功能單元上運行能夠在特定功能單元上運行的碼。

基本上，當一主要邏輯處理器上的緒實際正於工作者邏輯處理器上運行一運算流時，主要邏輯處理器並無任何事可做。而在作業系統認爲它正進行所有工作時有效停機。如上所詳述，作業系統並不了解工作者邏輯處理器。當一運算流在工作者上運行時，其專有的主要邏輯處理器正等待二件事之一發生：中斷或是運算流中的流退出。若這些事任一發生時，則主要（邏輯處理器）接管。若有移至中斷流的中斷時，則立即執行此中斷流，與在工作者上的其運算流並行。在大部分情況中，中斷流進行整個中斷操作並且主要（邏輯處理器）回至等待，對運算流沒有任何影響。當運算流中有流退出時，等待中的主要（邏輯處理器）建構該點的暫存器狀態，並且重返原始碼中。理想上，將會很快地命中一熱碼入口點。這將會開始執行一新的運算流、或者可能是在主要邏輯處理器上的一系統流。

DTSE 軟體 311 包括可能包括在詳述於後的其它任務之間用於產生、維持、及移除流的常式。

硬體「包整程式」

在一些實施例中，硬體包整程式用於動態緒切換執行邏輯。包整程式硬體支援一種以上後述功能：1）偵測熱區域（熱碼根偵測）；2）產生將以熱區域爲特徵的資訊（概析）；3）執行異動時對狀態予以緩衝；4）於成功的情況提交此緩衝狀態；5）於放棄的情況廢棄緩衝狀態；6）偵測連貫事件，諸如寫－寫及讀－寫衝突；7）防範交

互修改碼；及/或 7）防範分頁相關改變。各功能將詳細討論於後或是已經討論過。

當於 DTSE 執行模式時，產生的緒一起運作，但不具有溝通方式。當提交/時距標誌標明相當於原始碼中的一些 IP 的緒碼中的 IP 時，各核心執行其自身的緒。DTSE 硬體緩衝載入及儲存資訊，避免任何成爲外部可見的儲存（全域提交）。當雙核心已達到一時距標誌時，對載入及儲存予以查對違反。若未偵測到違反，此些儲存能夠成爲全域提交。此些儲存的提交標明一查對點，執行於此在違反/髒退出的情況須跳越。

圖 4 繪示依據一些實施例的用於包整程式的主要硬體區塊。如上所討論，這由二個以上的核心 401 組成（所示爲一成對關係，但可以有其它種）。違反偵測、基元性提交、熱 IP 偵測、及概析邏輯 403 結合於核心 401。於某些實施例中，此群組被稱爲動態緒切換執行硬體邏輯。同樣結合於這些核心的是中階快取 405，此處合併有第二執行模式之執行狀態。此外，有一末階快取（last level cache）407。最後，有一 xMC guardata 快取（XGC 409），將配合圖 7 詳細討論。

一些應用程式的「流」會由根偵測硬體中的硬體所識別（未明確繪出）。各流爲具有一個以上入口點的明確界定的位元碼組。應用程式的流組須包含有效量的應用程式之動態執行，但靜態流之結合大小由於流之平行化而必須爲小。

當原始碼正執行時，根偵測硬體檢索一單 IP 值，稱爲「根（root）」，會識別一新流。典型地，當執行於一流中時，此硬體將不檢索。在一些實施例中，硬體會持有一份 64 指令指標（IP）的清單。此清單依區位（location）0 至區位 63 排序，各區位能夠具有一 IP 或爲空。此清單由全空開始。

若有匹配在區位 N 的清單中的入口的 IP 的一合格分支（branch），那麼區位 N-1 與 N 互換區位，除非 N=0。若 N=0，則不發生任何事。更簡言之，於清單中此 IP 上移一個位置。

若有未匹配清單中的入口的 IP 的一合格分支，則入口 40 至 62 降 1 移位至區位 41 至 63。區位 63 之先前內容則失去。在區位 40 輸入新 IP。

在一些實施例中，對 IP 爲「適格」以加入清單或是「適格」匹配有限制，並因此而造成 IP 已經在清單上。這類限制第一爲僅採取返回分支之目標爲適格。呼叫及返回爲不適格。若採取返回分支正執行「熱」作爲部分流並且並未離開流，那麼它的目標爲不適格。若此採取返回分支之目標命中於熱碼入口點快取，它爲不適格。基本上，已經在流中的 IP 不該置入清單。

在一些實施例中，具有可有軟體設定的二個「排除」區域。各區域由一下限與一上限對排除區域的 IP 予以描述。注意此功能能夠設定僅對某些區域中的 IP 接收。第二限制爲排除區域中的 IP 爲不適格放入清單中。

在一些實施例中，少於 16,384 個動態指令的指令無法在命中清單中的指令後適格加入，然而，可允許將命中清單的最近 IP 取代爲 16,384 個動態指令窗內的一新 IP。基本上，一流以動態地平均最少 50,000 個指令爲目標。清單中的一 IP 對這樣的一流而言爲一潛在根源。因此之後的 16,000 個動態指令則視爲已提供於清單中的流之部分。

在一些實施例中，硬體保持 16 深度的堆疊。一呼叫循環地增量堆疊指標以及一返回減量堆疊指標，但不會環繞。亦即，在呼叫時，堆疊指標始終爲增量。但有一推入深度計數器。而無法超過 16。若會令推入深度爲負，則返回不使堆疊指標與推入深度計數器減量。每一指令使堆疊中的所有區位增量。在推入時，清除堆疊之新頂端。堆疊區位在 64K 的最大計數飽和。因此，另一限制是除非堆疊之頂端飽和否則沒有 IP 適格於加入清單。其理由在於避免錯誤迴圈。假設有一程序包含一總是重覆二次的迴圈。該程式會被各處的碼所呼叫。然後此程序中的返回分支會時常命中。雖然其檢視（look）非常頻繁，但是其爲來自各處的邏輯無關工作，因此將不會引導出好的流。在呼叫此者的程序中的 IP 爲所要者。最好是外部程序，而非內部者，除非內部程序是足夠包含一流的大小。

在一些實施例中，若一 IP，I，被加入清單或是晉升（由於命中一匹配），則之後 1024 個動態指令內不會有適格以匹配 I 的指令。此規則的目的在於避免高估緊密迴

圈。在這樣的迴圈中的返回分支會大量命中，但每個命中並不代表有更多的工作。

清單中的首位 IP 視爲代表非常活躍的碼。

典型的工作負荷將具有一些流以取得高動態涵蓋。雖然爲了提早得到最大的性能增益，最好是大致上依重要性來產生這些流，但並非必定是完全依重要性來得到。應得到建立流的合理位置。這會成爲熱碼，並且超出找出接下來所從事流的行事。最可能地，會得到大量的流。

流，一般而言，並不互斥且可能部分重疊。但是，至少各流之根不在先前所得到的流中。事實上可能仍然在之後所得到的流中。這足夠證明不會有完全相同的二個流。儘管在之前已經使用特定數，這些僅僅是舉例。

有用於這種架構的至少二種流：系統流以及運算流。運算流在工作者邏輯處理器的叢集上執行。系統流在主要邏輯處理器上執行。

運算流可爲，但非必須，藉由緒分解的單碼串流所構成的多重軌跡。運算流完全由單一叢集內的工作者邏輯處理器執行，因爲整個流在查對後必須依程式次序而全域提交（但典型地爲全域提交爲一連串的提交時距，而非單一片斷）。所有執行流的工作者能夠自由載入及儲存於相同位址。運算流最受何者不能做的項目所限制。

系統流也是藉由 DTSE 軟體來產生碼並存在隱藏記憶體中。有爲數眾多的事件無法在系統流中完成。其受限，但比運算流受到較少限制。系統流爲運算流之間的一序列

碼或是於運算流的一獨立平行緒。

系統流的一個重要用途在於中斷處理。有時，DTSE軟體創造流，其入口點爲最普遍的中斷向量目標。此碼將涵蓋對於這些向量目標最普遍的路徑。若中斷能夠被完整控制於系統流內，自向量目標至中斷返回（IRET）返回中斷的碼無需系統流退出，則無需干擾執行該「中斷的」流的工作者。若整個中斷處理序列無法如此處理，則在執行該「中斷的」流的工作者中會強制流退出。上下文交換總是會造成中斷流退出，且因此爲該「中斷的」運算流中的流退出。

若中斷能夠被控制在此中斷流中，則暫存器則不會儲存。中斷系統流爲一單全域提交時距。若非會結束以IRET及全提交爲一單位，則是將狀態恢復至中斷向量之目標以及自原始碼中該處重返執行。狀態恢復包括停止運算流中的全域提交（與進一步執行），並且自運算流中最近的全域提交點來恢復暫存器狀態。

隱藏記憶體中的流中的碼能夠特別快又有效地被緒切換，因爲緒切換被確實地程式化寫入該碼。原始碼無法具有被程式化寫入的緒切換。涉及原始碼的緒切換既緩慢又無效率。

運算流被限制何者能夠做。一些原始碼並不適格以放入運算流中。一些非適格於運算流的碼能夠放入限制較少的系統流中。仍然有一些即使放入系統流中卻不適格的碼。必須仍爲原始碼。

在一些實施例中，若是需要包括非限量的原始碼，則以二類型之流而多工處理。

一旦測得熱碼根 IP（入口 IP），則提供主要核心以便在 IP 下次命中時，該核心將開始概析此原始碼。在概析時，上述資訊（分支、載入及儲存）會依程式動態順序儲存在記憶體中的緩衝區。這些緩衝區在之後會由緒產生軟體使用來指示緒產生一消除非使用碼（基於分支）、指示最佳化、以及偵測載入/儲存關係。在一些實施例中，用於衝突查對（將說明於後）的相同硬體用於自引退而緩衝載入、儲存、及分支資訊，並且將其泄入記憶體。在其它實施例中，微作業在將需要資訊直接儲存於記憶體的執行時插入程式中。

要區別熱區域的特徵（概析），緒執行模式軟體需要下列一個或多個資訊：1）對於分支，需要 a）帶有或未帶有資訊的條件分支、以及 b）分支目標的間接分支；2）對於載入，a）載入位址及 b）存取大小；以及 3）對於儲存，a）儲存位址以及 b）儲存大小。

在一些實施例中，序列緩衝區（OB）會維持用於概析。這是因爲載入、儲存、及分支爲執行非序（out-of-order），但概析資料需要定序。OB 與重定序緩衝區（ROB）相同大小。當配送時，載入執行會將它們的位址及大小寫入 OB。在 STA（儲存位址）配送期間，儲存執行將進行相同動作（STA 配送優於儲存引退，配送之目的在於將虛擬儲存位址轉移爲實體位址）。分支將寫「對象」欄位，

而能夠用於直接及間接分支。當這些載入、儲存及分支自ROB引退時，其對應資訊將會自OB處複製。熱碼概析使用包整程式硬體能夠緩衝異動狀態並於之後提交的事實。它會使用相同的提交緩衝狀態之資料路徑以自OB複製資料到一寫入組合快取（將於之後說明），然後提交。概析資訊將會寫入之後會被緒執行軟體所使用到的特有記憶體區位中的一專屬緩衝區。

在一些實施例中，DTSE軟體將IP寫入一暫存器並將其提供。硬體將會取得概析資料並且將其在IP命中時寫入記憶體中的一緩衝區。在流根IP之後所遭遇的某些數量的分支（例，10,000次）的分支方向歷史紀錄會由硬體於執行期間回報。該清單係依局部引退次序而爲每個分支一個位元。動態緒切換執行軟體在引退時得到將爲分支之目標。其報告嵌入分支方向串流的間接分支之目標。在同時，硬體將回報載入與儲存的位址及大小。

隨著應用程式可能隨著時間改變行爲，執行應在應用程式執行期間維持監控。當流呈現出變得「太」短，應該對其增長或偵測。加之，新流能夠在任何給定時間發現、再生、及合併。在一些實施例中，動態緒切換執行系統的硬體將會回報對各流的平均全域指派指令及循環。若有任何疑問時，軟體將會需要對此考慮並且也偶爾在原始碼取得資料，藉由暫時地對流失能（disabling）。在大部分的範例中，軟體並不運行「熱」碼除非相當清楚是淨勝（net win）。若不清楚「熱」碼爲淨勝，軟體應該對其禁能

。這能夠一個接一個流完成，或是軟體能夠只將關於此工作負荷的整個事件關閉。

軟體將會接續接收分支失誤預測資料以及分支方向資料。此外，軟體將會得到關於緒停滯（stall）的報告，因爲其在全域佇列之區段額滿、或等待流完成。這些能夠暗指運行超前太多的一待載入軌跡（討論於後）。也將得到快取失誤的核心停滯時間。例如，取得大量快取失誤停滯時間的核心A能夠解釋爲何核心B運行超前太多。這全部能夠用於對此流進行軌跡的較佳載入平衡。硬體將同時會回報具有最高快取失誤率的載入之完整識別。這能夠幫助軟體對快取失誤進行再分配。

在一些實施例中，軟體將會得到每個流執行中的循環或指令的報告。這將會識別太小的流，並因此爲極度的覆蓋過載。

圖5繪示根據一實施例的時距執行。當緒執行軟體產生熱碼的緒時，其試著伴隨盡可能少的重覆而進行。由於原始靜態碼，創造出二個以上的緒。這些緒爲時距的。產生時距標誌同步於原始碼並且在時距標誌邊界處查對違反（諸如上文所言）。記憶體狀態可於每個時距的完成時提交。如圖所示，當命中熱IP時，執行模式切換至緒執行。與以前一般圖式相異之處在於各緒具有時距。在各時距後，做一查對（chk）。在範例中，在第二時距執行之後，查對（chk2）找到一違反。由於此違反，碼退至最近的查對點（其可在緒之後或是命中熱IP之前）。

如上所述，當熱碼區域退出（淨退出）或是在違反情況（髒退出）時，緒執行模式將退出。在淨退出時，此退出點將標明一時距及提交點，以提交所有儲存。在淨與髒退出二者的情況時，原始碼將移到最近的查對點（提交）之對應原始 IP。暫存器狀態將必須從兩核心之狀態來合併。對此，緒產生器將必須在各提交更新暫存器查對點資訊。這能夠例如藉由插入會自各核心將相關的暫存器存入硬體緩衝或記憶體的特有儲存而完成。在退出時，暫存器狀態將自兩核心合併入原始（主要）核心。應注意存在暫存器合併的其它選擇退出，例如暫存器狀態可檢索自緩衝的載入及儲存資訊（如在產生時藉由緒產生器來決定）。

DTSE 硬體之一實施例的一更詳細圖式繪示於圖 6 中。這圖式描繪推測執行於左以及一致性於右。在一些實施例中，除了 MLC（中階快取）617 與核心 601 以外的每一東西形成一部分的違反偵測、基元性提交、熱 IP 偵測、及概析邏輯 403。此執行爲推測性，因爲當在緒化模式中，核心 601 中產生的緒一起運作，它們並不互相溝通。

在一些實施例中，硬體包整程式包括一單一實體保護快取，其利用在保留記憶體（同樣地，並未繪示出）中所擁有的副本（以某種形式）來保護原始碼。在一些實施例中，有 16K 線、8 路集合相聯快取，其中每條線適用 4096 位元組的原始碼。解析度是按 64 位元組區塊。快取線在線範圍排列的 4096 位元組內含有一開始區塊以及一結束區塊。一區塊數爲 6 位元。此快取在同樣標籤的相同集合

中可具有多重線。因此，當讀取此實體保護快取時，相同集合中的多重線可能有 8 線命中。其提供在相同 4096 位元組中表現數個不相交時距的能力，在之間具有無保護的間隙。

軟體 311 應保證 64 位元組區塊沒有表現在實體保護快取中多於一次。當做出對命中 64 位元組區塊的查對時，對於該區塊是否爲在 4096 位元組間隔位址命中的任何最多 8 線的查對爲被執行。由於有 16K 線，則編號以 14 位元線號碼。64 位元組區塊爲錯失此快取或是命中唯一 14 位元線號碼。

實體保護快取在任何失效窺探時被查詢。命中時，檢索該命中之 14 位元線號碼，並且選擇性提供命中的 64 位元組區塊之 6 位元號碼。

對於各流，軟體將會保證其原始碼在處理之前會被涵蓋於實體保護快取中。這可能涉及擴展已在快取中或加入新線的線之時距。

軟體將會得到需要涵蓋於流的線號碼之清單。例如，軟體將會減少此清單至一單一 28 位元號碼。該線號碼爲 14 個位元。對線號碼中的各位元使用 2 個位元，有效位元及資料位元，因此使用 28 個位元於 14 位元線號碼。在各位元位置中，若整個清單對該位元同意，該位元成爲結果的資料位元且有效位元變爲 1。若在清單中對一位元位置中的值有異議，則其結果的資料位元變爲 0 以及有效位元變爲 0。這 28 個位元爲該流之標記。

各核心具有在其熱碼入口點快取中的所有流的標記（圖未示）。並且，特別地，其具有當前暫時地全域提交流之一個標記。在保留記憶體中具有熱碼入口點表，雖然在一些實施例中，該表的部分爲被儲列。

若預料到一引用分支，則在最寬範圍快取中查詢其目標。若失誤，則請求重填。若命中並且沒有入口點，則爲其終止。若有入口點，則查詢次一較小範圍快取。最終會是命中一入口點、未命中一入口點、或是不確定但是取得於該快取的重填。

若該引用分支目標在保留記憶體中，則沒有熱碼入口點的快取尋找。

取得引用分支的一識別。若命中一入口點，則當此分支引退並且仍被引用時，強制跳躍至指示目標，並且工作者同樣強制跳躍至該指示入口點。

查對各暫時全域提交流，若有線匹配，則該處將會是一流退出。否則允許其提交。這持續直到在熱碼入口點快取清理之前進入的所有流完成。一旦此查對適當，能夠重返全域提交。

必須查對熱碼入口點快取中的所有標記。若有線匹配，則撤銷該入口點。當驗證碼被調用時，將會查看旗標。這表示在能夠驗證任何入口點之前，必須處理保留記憶體代理（圖未示）中的工作清單。讀取被命中的線號碼。得知各線號碼的確切時距。確切得知各流所請求的快取線。刪除或核對可能已經命中的各流。更新其整個熱碼入口點

表。然後能夠進行其驗證任務。

當時距標誌標明對應於原始碼中相同 IP 的緒碼中的地點（IP）時，各核心 601 執行自身的緒（時距標誌顯示在前面的圖示中）。當執行於此模式時，硬體緩衝載入與儲存資訊，避免任何儲存爲外部可見（全域提交）。此資訊可儲存於各種快取中。這些繪示爲推測載入資料快取（SLC）603、載入儲存序列緩衝（LSOB）605、以及推測儲存快取（SSC）607。雖然這些繪示爲獨立，在一些實施例中，它們是單一實體的分割。加之，此儲存器可不依據如圖般的每一核心準則而配置。

在一些實施例中，在抓取這些資料之前，通過如先前詳述般維持執行載入及儲存的適當引退順序的序列緩衝器（OB）602。使得實體位址及儲存之大小有效於 DTSE 邏輯，如同將高位（較早的）儲存寫至資料快取並且置於 SLC 603 中。SLC 603 同時偵測失效窺探。相似地，使得載入位址及大小有效於 DTSE 邏輯並且儲存於 SLC 中。載入與儲存同樣寫至載入儲存序列緩衝（LSOB）605 以保持各緒中的載入與儲存之間的資訊成序。LSOB 605 維持該資料及一遮罩（有效位元組）。

最終，DTSE 邏輯取得依與其儲存位址的適當序列的引退載入位址。在一些實施例中，DTSE 邏輯在執行時間取得該載入位址及大小。若是那種情況，DTSE 邏輯將會老化並且將它們過濾以取得適當序列，引退載入位址及大小。

資料快取

當兩核心到達一時距標誌時，則使用如詳述於後的違反查對部件（儲存正確性快取（SCC）611 與載入正確性快取（LCC）613）來查對違反的載入與儲存。典型地，每個工作者有一 SCC 611 及一 LCC 613。SCC 611 寫入其工作者的載入與儲存位址。其與所有其它協作工作者之儲存位置而一同讀取。其顯示在單一調準時距中已經寫入碼的位元組。典型地，這情況爲一獨立實體結構。LCC 613 被寫入來自 LSOB 605 的所有工作者之儲存位址。其與其工作者之載入位址一同讀取。典型地，這是保留記憶體中的資料結構。若未偵測到違反，該儲存能夠成爲全域提交。儲存之提交表示一查對點，其爲當在後續時距（spans）而有違反時執行應該跳至之處。

有數種可能發生的違反。其一爲來自外部實體（例，另一核心）的失效窺探，此外部實體使得由核心群其中一個核心所使用的資料無效化。雖然有些值爲假定的（推測執行），可能會錯誤，該執行必須放棄並且原始碼將返回至最近的查對點。當相異緒的兩個儲存係於相同時距寫至相同位址時，則可能引起儲存/儲存違反。在一些實施例中，由於在單一時距的相異緒之間沒有成序，無法得知在原始程式序列中哪個儲存在後，因此緒執行模式放棄並且返回至原始執行模式。若相異緒中的一儲存與一載入於相同時距在記憶體中使用相同位址，則可能引起儲存/載入

違反。由於在緒之間不存在溝通，該載入可能錯失由該儲存動作所儲存的資料。應注意，典型上，一載入不被允許命中在任何過去時距中關於其它核心的一儲存資料。那是因爲核心獨立執行，並且在其它核心到達該儲存之前（任一核心能夠超前其它核心許多時距），該載入可能已經執行。自我修改碼或交互修改碼事件可能發生，其中原始碼已經由程式中的一儲存或是一些其它代理（例，核心）修改。在這種情況，緒碼可能變得陳舊。其它違反可能因性能最佳化以及架構交換而發生。這類違反的範例爲一 L1 資料快取單元錯失命中下降推測儲存（若不爲硬體所支援）。另一範例爲一假定由緒產生器所製造，其知後偵測爲錯誤（斷言硬體區塊 609）。

一旦保證沒有違反發生，經緩衝之儲存可被提交並且變得全域可見。此基元性地發生，否則可能破壞儲存序化（儲存序化爲記憶體序列架構的部分，處理器必須遵守）。

當執行緒模式時，所有儲存將不使用「規則的」資料路徑，而是將都會寫至（執行該儲存的核心之）第一階快取，其係作爲私用、非一致性圖刷暫存區（scratchpad），以及至專屬資料儲存器。資料儲存器（快取及其上的緩衝）中的資訊將包括該儲存之位址、資料、及資料大小/遮罩。當儲存爲來自相同提交區域時，則允許儲存結合。

當硬體決定提交狀態時（在已經查對到違反之後），需要自資料儲存器（例，SSC 607）排出所有儲存並且變

爲一一致性、可窺探的狀態。這藉由移動該儲存自資料儲存器至一寫入結合快取（WCC）615 來完成。在資料複製期間，窺探失效將會送至所有其它的一致性代理者（agent），因此該儲存將在其改變的快取線上取得所有權。

寫入結合快取 615 結合來自相異代理者（核心及緒）的儲存，在相同最佳化區域工作，並且使得這些儲存變爲全域可見狀態。一旦來自核心的所有儲存結合入 WCC 615，其變爲可窺探。這提供維持記憶體序列規則的基元提交於一中階快取 217。

藉由清理資料儲存器中的一些「有效」位元而使緩衝狀態廢棄於一放棄中，藉以移除所有的緩衝狀態。

一致性查對可能因原始程式被分割爲二個以上的並行緒而被使用。若該軟體最佳化器不正確地消除載入及儲存，則可能發生錯誤結果。使用接著的硬體建立區塊查對讀－寫以及寫－寫衝突。載入正確性快取（LCC）613 保持最佳化區域中所執行的載入之位址及資料大小（或遮罩）。用以確保不會有來自另一邏輯核心的儲存與來自該最佳化區域的載入相抵觸。在時距違反查對上，各核心將其儲存寫入其它核心之 LCC 613（對該核心寫入的各位元組設定一有效位元）。LCC 613 接著保持其它核心之儲存之位址。各核心藉由反覆載入覆蓋其 LSOB（載入儲存序列緩衝）605、重置其儲存所寫的各位元組的有效位元、以及查對並未命中具有一有效但卻設爲 1 的位元組（意指該位元組是被其它核心寫入）的各載入來查對自身的載入。命

中有效位元 1 的載入代表一違反。一儲存正確性快取（SCC）611 保持在最佳化區域中執行的儲存之位址與遮罩。來自這個快取的資訊與協作邏輯核心之 LSOB 605 中的入口比對，以確保沒有未被偵測到的衝突。時距違反查對時，SCC 611 爲重置。各核心將其儲存自其 LSOB 605 寫至其它核心的 SCC 611。然後各核心將其儲存（來自 LSOB）與已經在其它核心的 SCC 611 中的儲存相查對。若一儲存命中來自它處的儲存，則偵測到違反。應注意一些儲存可能被緒產生器複製。必須藉由 SCC 611 正確地操作這些儲存，以避免錯誤違反偵測。此外，推測載入快取（SLC）603 防範來自最佳化區域的載入對於來自在緒執行規劃描述下並未協作的邏輯核心的窺探失效，但是可能並行運行相同應用程式之其它緒或是存取共用資料。在一些實施例中，在此所述的緒執行規劃實行一「全有或全無」策略並且在最佳化區域中的所有記憶體異動應視爲如同及時（提交時間）在單一點全部一起執行。

當在 DTSE 執行模式中時，各核心之 L1 資料快取運作如一圖刷暫存區。儲存應不對窺探作出反應（以避免任何儲存被全域可見）並且推測資料（非查對/提交資料）應不寫回中階快取。在退出這個模式時，應自資料快取廢棄可能退卻的所有推測資料。注意，由於一些實施交換，在緒執行模式時，可能需要使已經執行的所有儲存或載入資料失效。在資料快取中，原始程式資料區塊、以及所有全域提交點的軌跡資料區塊都會儲存於相同位置。它們都

由該區塊之原始程式位址所定址。

特別要注意上述範例輕易地廣義包括多於二個核心協作於 DTSE 執行模式。並且一些違反可由硬體藉由核心執行停滯或同步而暫時解決（例，一些載入/儲存違反）。

在一些實施例中，並不在每一時距完成提交。在這種情況中，將在各時距完成違反查對，而提交將是每少數時距完成一次（以減少暫存器查對點過載）。

在一些實施例中，資料快取及其入口非是特有的。隨著支援的工作者量，資料快取需要一些以位址空間地圖形式的幫助。例如，各資料快取標籤可擴展具有一 x 位元位址空間地圖（諸如假設每核心 4 個工作者則 5 位元值，若每個核心僅 2 個工作者則 3 個位元，或是若每個核心僅有 1 個工作者則 2 個位元）。

在 5 位元變數中，位元 4 代表是否主要邏輯處理器能夠存取此快取線。各工作者邏輯處理器具有一位元於該區 <3:0>。這代表是否相關的工作者邏輯處理器能夠存取此線（例，0001 意指工作者邏輯處理器 1 能夠存取該線而其它則不行）。工作者邏輯處理器在位址空間地圖中的它的位元未設定時會取得一資料快取失誤。主要邏輯處理器在位址空間地圖中的主要位元未設定時會取得一資料快取失誤。

資料快取可含有多重線於相同位址（於多路），假如每次存取會命中（最多）其中之一。這意指對於在位址空間地圖中的 5 個位元中的每一個，在相同位址上於使該位

元被設定下的資料快取中不會超過 1 線。

當（具有相同標籤的多路的）資料快取的讀取時，將會選擇請求邏輯處理所可存取的唯一的路，若任一存在時。若沒有，則爲資料快取失誤。

在高階儲存處理（最原始儲存的處理）時，高階儲存可能只進行可存取於執行儲存的邏輯處理器的快取線。再者，若工作者寫入一高階儲存至資料快取，則目標線取得設定爲僅可存取於進行該儲存的邏輯處理器的它的位址空間地圖。所有主要邏輯處理器對此目的是相同的。

加之，在一些實施例中，有邏輯相關於各資料快取線的一「現行位元」。其並不必須實際上接近資料快取或是資料快取標籤。現行位元，對於工作者邏輯處理器可存取的所有快取線，在工作者邏輯處理器中的全域提交標誌處於高階之時被清理。於該線的任何成功寫入設定爲「現行」。於該線的寫回至中階快取設定爲「現行」。當來自中階快取的重填時，若在該中階快取中爲髒的且非現行時，該線必須設定爲「非現行」。除此之外，於該資料快取中設定現行。即使沒有資料，基於來自該中階快取的回應，現行位元以此方式設定；MLC 正好授與所有權。

中階快取形成髒且非現行並非一件平常的事。這件事會送回至資料快取並且設定資料快取中現行位元的狀態，即使沒有資料；MLC 正好授與所有權。

在儲存放入資料快取之前，工作者將高階儲存寫至一使其現行位元停止的髒資料快取線的試行引起寫回，但並

非一失效。進行資料快取線的寫回設定其現行位元。並且成功寫操作該高階儲存，若適當則（在寫回後引起）設定該線「現行」。

若一高階儲存的寫操作命中一使其現行位元停止的淨資料快取線（E 狀態），中階快取則發訊（若爲髒的）以將其關於此線的備份寫至最近階的快取，並且取得關於一新的軌跡資料區塊的一新的實體位址。當髒資料目前進入資料快取時，中階快取中的髒資料不會在相同全域提交時距中進行。資料快取線取得現行作爲將該高階儲存寫入的結果。

在一些實施例中，有一元資料緩衝器（圖未示），用於各工作者邏輯處理器。元資料連續地寫至緩衝器。元資料緩衝器應要能夠將這資料寫至中階快取。軌跡執行導致軌跡資料寫至保留記憶體並且元資料寫至保留記憶體。處理的剩餘部分爲，自保留記憶體讀回元資料，查對它，然後合併自保留記憶體讀取的各種軌跡的軌跡資料，以及使其如原始程式位址空間的狀態般可見。

在中階快取 217 中，原始程式資料區塊，以及關於所有全域提交點的軌跡資料區塊被儲存在相同位置。它們都以該區塊的原始程式位址來定址。

在中階快取，於標籤有一擴展，保持中階快取線的現行真實實體位址。真實實體位址爲自中階快取寫回至最近階快取所需。其不爲中階快取中大部分的其它作業所用。在一些實施例中，真實實體位址爲軌跡資料區塊號碼。其

不會大於 16 位元。

在一些實施例中，與中階快取相關的硬體暫存器保持基址以及用於各工作者邏輯處理器的軌跡資料空間的限制，以及各軌跡的最近配置的軌跡資料區塊。此硬體同時保留現行全域提交號碼。

有一邏輯相關於各中階快取線的「現行位元」。並非必須實體接近於中階快取或中階快取標籤。這個位元，用於所有的快取線，是工作者邏輯處理器可存取的，在一全域提交標誌於此工作者中進階時清除。

在一些實施例中，各中階快取標籤擴展有一 6 位元（假設支援每核心 4 個工作者）位址空間地圖。這個標籤的大小取決於每個核心的工作者數量。

位址空間地圖支援在同時於相同位址（包括相同標籤）於中階快取中具有多線。其載明哪個邏輯處理器能夠存取該線。在中階快取之讀取時（相同標籤的多線），將會選擇請求邏輯處理器可存取的唯一路徑，若任一存在。若無，則爲中階快取失誤。

在一些實施例中，中階快取中各線可具有一暫時性提交位元以及一暫時性除役（decommissioned）位元。

提交將會由清理真實實體位址上的有效位元以及設定主要可存取的快閃構成，用於所有暫時性提交快取線以及對所有暫時性儲除役快取線清理線有效位元的快閃。

在一些實施例中，對於各 DTSE 叢集有一些查對狀態機器。各查對狀態機器可從事於一緒，故所提供的狀態機

器的數量決定叢集能夠同時從事的緒的數量。查對狀態機器自所有從事於相同緒的軌跡讀取元資料清單並且透過儲存正確性快取與載入正確性快取來對它們處理。也寫出一複製回復清單。

在一些實施例中，複製回復狀態機器分配出自複製回復清單的快取線至具有中階快取的合併狀態機器以進行不同軌跡儲存資料的實際合併以及將結果置入原始程式位址空間。

在一些實施例中，每一中階快取恰好有一個合併狀態機器。在必要之時，這個狀態機器進行出自數個軌跡之軌跡資料儲存器的資料的實際合併。

一旦執行一載入，必須可證明，除了執行該載入的緒之外，系統中沒有經代理的儲存能夠命中這個載入，直到該載入被全域提交。必須有出自任何載入或儲存的核心引退的保證直到其全域提交（一些協作於這個緒的執行中的中階快取）已具有至少在共用狀態中的目標原始程式位址線。此線能夠通過協作中階快取之間附近（主從方法）但必須維持連續持有。這意指對於取得這線的所有權的任何其它代理，當沒有其它協作中階快取持有該線時，至少一個協作中階快取將會取得一失效窺探。

典型地，各 DTSE 叢集具有一持有快取。這個結構表示能夠用於確信叢集中的一些中階快取的快取線將得到來自環或資料快取的窺探。典型地，持有快取位於保留記憶體中，但一部分可能貯藏有查對者。

各工作者邏輯處理器具有一推測存取快取。推測存取快取代表已經由工作者邏輯處理器局部引退的載入及儲存，但是尚未全域提交。

全域提交標誌是對於特定位址的一儲存。資料給定一分類號以及對提交指標描述符的一指標。

提交等待是對於一組特定位址的一載入。能夠出現在任何地方。取決於該組中確切的位址，能夠是等待在最近全域提交點、或是一確信分類的最近全域提交點的提交，或是其它可能選擇。這個載入的位址能夠請求最近全域提交記憶體狀態。

DTSE 不會回應於一提交等待，並因此保持在停滯的請求工作者中的執行直至發生具體的全域提交。若被請求最近全域提交記憶體狀態，DTSE 邏輯將進一步使請求工作者中的所有軌跡失效。然後 DTSE 邏輯將返回資料供提交等待。這將使此工作者非停滯並且允許其繼續進行。

若最近全域提交記憶體狀態被請求時，所有在等待所指定的全域提交點之後的此工作者中的所有儲存將會遺失。一般而言，實際上只有等待全域提交點之後使用這選擇，並且在這個工作者進行任何額外儲存之前。典型地，等待將會緊接於所等待的全域提交點的全域提交標誌之後。

等待有三種用途：等待能夠用於防止暫存器狀態被改變，因此迴避保留一些暫存器的必要性；當一載入有時命中不同軌跡的儲存時，提供該載入取得正確結果的能力；以及能夠用於避免脫控推測。

II. 運作

資料記憶體狀態由資料本身以及元資料所表示。不僅是個別地並且於不同處儲存。主要邏輯處理器以及工作邏輯處理器皆具有元資料。在缺少元資料的情況，記憶體狀態無法有效於工作者邏輯處理器。元資料僅用於主要邏輯處理器的概析。

至於保留記憶體中的資料空間，在一些實施例中，各軌跡於保留記憶體中具有自身的資料空間，如各工作者邏輯處理器。主要邏輯處理器不具有軌跡資料空間。

在進入流時，所有軌跡資料空間爲未分配。在執行流時，快取線尺寸區塊視需求循環連續地分配。各分配用於一特定全域提交時距。當所分配的時距被全域提交時，區塊依序解除分配。對於各工作者邏輯處理器有一分配指標以及一解除分配指標。這些指標含有區塊號碼，而非位址。區塊號碼空間將是存在於這個分配的實體區塊的數量的乘積。

於分配中，區塊號碼無法包整程式。當所有區塊號碼已經分配過一次時，這個軌跡無法再多的分配軌跡資料區塊。若需要另外的分配，將會造成流退出。

另一方面，實體區塊的分配能夠包整程式多次，直到未包整程式區塊號碼的界限。於真實實體位址具有解除分配區塊號碼的中階快取中的線不會寫回至最近階快取。相似地，若在中階快取中有失誤並且重填的真實實體位址爲

一解除分配區塊號碼，重填將會出自原始程式位址替代。這樣的線之中無法具有推測儲存。若有推測儲存，則應已經在最近重分配。

可以看出應要有足夠實體區塊用於軌跡資料的分配以涵蓋正執行的碼中的邏輯位置能夠相異的軌跡總數，外加一些涵蓋查對及全域提交的時間。區塊號碼空間必須涵蓋流自開始至流退出的整個執行。

若軌跡中有一狀態恢復，工作者執行該軌跡的軌跡資料空間則初始化爲空的，加上用於分配的原始開始點。

當軌跡執行時，邏輯關聯於工作者邏輯處理器的分配發生。複製回復狀態機器將這空間中的特定位置與之後在全域提交處理中的儲存關聯在一起。

複製回復狀態機器將各合併狀態機器所正使用的一合併工作佇列寫入記憶體中。其描述合併狀態機器應存取的軌跡資料區塊。由於會通過記憶體，故應保持可能的壓縮。

各邏輯處理器的元資料爲個別儲存。在一些實施例中，邏輯處理器的元資料爲程式正確位址、大小及載入/儲存指示值的一線性陣列。其含有校正標誌以及全域提交標誌還有複製儲存標誌。亦可含有分支方向及間接分支之目標。其亦可紀錄在中階快取所接收的失效。

當核心載入於資料的原始程式位址時，硬體將抓取位址及大小並且將其加入元資料清單。若工作者自主要可存取且擁有的資料快取線載入，若是髒的，則（若便於此設

計）在強制寫回中階快取之後使其共用。若並不方便，則能夠使非主要可存取且非現行的線作爲替代。任一這些動作將確保由非與此工作者協作的邏輯處理器於此線的任何未來的儲存的通知。

於執行模式中出自工作者的高階儲存不需要資料快取中的所有權以寫該資料快取。高階儲存能夠寫至共用狀態的一線，但該線必須軌跡可存取。在任何事件中，若出自Catalina 工作者的高階儲存被寫至一資料快取線，線變成僅爲那個工作者可存取。無論那線之前如何，已經轉換成軌跡資料。注意，若該線爲髒的，在一些實施例中，在該高階儲存寫操作之前，可能已經強制寫回，除非是已經軌跡資料且現行。

有因於此，當處理於那線的一高階儲存時，所有權不被請求資料快取線。

由於這確實是於軌跡資料的儲存，中階快取將不會需要或請求原始程式位址線之所有權。其將具有軌跡資料線之所有權。

這是重要的，因爲其它軌跡可能正儲存至這線。合併引擎將需要取得原始程式位址線的所有權以進行該提交。其它中階快取能夠具有被共用且可由軌跡（非由主要）讀取的線，但無法具有其自身的。因爲在此所述的供應，不需要其自身的並且將不請求其自身。

這意指當一個中階快取正用於進行合併且正提交，並且請求原始程式位址線的所有權時，其它軌跡能夠保持那

線於共用狀態並且對其讀取且寫入（如所述般，對其寫入會將其轉換至軌跡資料）。

當資料快取失誤時，該線一般會自中階快取被請求。

當核心儲存至資料的原始程式位址，硬體將抓取位址與大小並且將其加入元資料清單上。

在資料快取中，原始程式資料區塊、以及全域提交點的軌跡資料區塊都位於相同地方。都是由該區塊的原始程式位址所定址。

現行位元，對於所有這個工作者可存取的快取線，在全域提交標誌處於高階之時被清理。於該線的任何成功寫入設定爲「現行」。於該線的寫回至中階快取設定爲「現行」。當來自中階快取的重填時，若在該中階快取中爲髒的且非現行時，該線必須設定爲非現行。否則設定爲現行。即使沒有資料，基於來自該中階快取的回應，現行位元以此方式設定。

在儲存放入資料快取之前，工作者將高階儲存寫至一使其現行位元停止的髒資料快取線的試行引起寫回，但並非一失效。設定一位元具有此寫回以告知此中階快取去分配一新軌跡資料區塊。進行資料快取線的寫回設定其現行位元。並且成功寫操作該高階儲存，若適當則（在寫回後引起）設定該線「現行」。

在儲存放入資料快取之前，可存取此線的工作者將高階儲存寫至一主要可存取的髒資料快取線的試行引起寫回，但並非失效。設定一位元具有此寫回以告知此中階快取

去分配一新軌跡資料區塊。進行資料快取線的寫回設定其現行位元。並且成功寫操作該高階儲存，若適當則（在寫回後引起）設定該線「現行」。

若一工作者已經存取這線，寫操作一高階儲存於主要可存取的乾淨的資料快取線時，中階快取若爲髒但無條件對新軌跡資料區塊取得新實體位址時，則傳訊以將這線寫回至末階快取。成功寫操作高階儲存設定該線「現行」。

若一高階儲存的工作者寫操作命中一使其現行位元停止的淨資料快取線（E 狀態），中階快取則發訊（有條件地，若爲髒的）以將其關於此線的備份寫至最近階的快取，並且取得關於一新的軌跡資料區塊的一新的實體位址。當髒資料目前進入資料快取時，中階快取中的髒資料不會在相同全域提交時距中進行。資料快取線取得現行作爲將該高階儲存寫入的結果。

若有一高階儲存以及有一非爲 E 或 M 狀態的線於資料快取中，則照例自中階快取對該線產生用於所有權請求的一讀取。

資料快取能夠要求中階快取去「寫回」一線並且分配一新軌跡資料區塊。若該線於中階快取中爲髒，則被寫回並且分配一新軌跡資料區塊。即使該線於中階快取中爲淨，該請求可能無條件地列入分配一新軌跡資料區塊。若一新軌跡資料區塊被分配，其取代在那位置的區塊。資料是相同的。

當所有寫回於中階快取時，資料快取「現行位元」（

在因寫回而被設定之前）被送至中階快取。資料快取寫回有二個原因：可能是資料快取中的一慣例驅逐或是一新高階儲存已命中一非現行的髒資料快取線。若一新高階儲存已命中一非現行的髒資料快取線，則資料快取送出一被斷言的位元。

中階快取將分配一新軌跡資料區塊。

在一些實施例中，於保留記憶體中有對於各軌跡的一快取（軌跡資料目錄）。在一線中的資料僅爲軌跡資料區塊號碼以及一全域提交號碼。此快取只有當一軌跡可存取但主要不可存取的髒軌跡資料區塊被驅逐時被寫，與分配一新軌跡資料區塊無關。

在一些實施例中，有對映於被稱爲軌跡資料目錄摘要的軌跡資料目錄的另一結構。若在工作者邏輯處理器有一中階快取失誤，則涵蓋其於軌跡資料目錄的位置的位元自該軌跡資料目錄摘要被讀取。通常會是無效的，指出在軌跡資料目錄中的失誤。在這個情況中，於中階快取的重填應來自區塊的原始程式位址。此資料可能已經在中階快取中做爲一主要可存取版本。在這個情況中，此版本能夠簡單地變爲工作者也可存取。若摘要顯示一命中的可能性，軌跡資料目錄必須被存取以取得最現行的軌跡資料區塊號碼，若結果是確實爲一命中。邏輯能夠將其轉換成區塊的位址。照例對於失誤，該區塊接著經由該環請求。

分配一新軌跡資料區塊是自中階快取至最近階快取的寫回的一特定形式。若中階快取意圖寫回一線至最近階快

取，並且此線爲非主要可存取的，但並不具有一有效的實際實體位址，那麼在該寫回能夠處理之前會分配一新軌跡資料區塊給它。該寫回前往由分配新軌跡資料區塊所得到的實體位址。分配一新軌跡資料區塊的處理在本說明書先前的章節討論過。

在每一軌跡中，執行將會命中一個別的全域提交點。有一儲存於一特定位址，此處做爲全域提交點標誌。其給予該新全域提交號碼增量。在其擁有的軌跡中的全域提交號碼增加。其保持爲關聯於中階快取的 DTSE 邏輯的一部分。如上述所提到的，這會造成儲存前往新資料區塊。並且關聯於資料快取與中階快取的「現行位元」被清理。

儲存僅在全域提交點變成全域觀測。載入及儲存爲全域推測，並因此在被一失效窺探命中時易受傷害，直至次一全域提交點。

軌跡設定一表示完成此提交時距的旗標。一查對狀態機器等待出自所有軌跡的此旗標。

如上所詳述，各工作者邏輯處理器具有一儲存正確性快取。此快取執行在相同調準時距中的一個工作者中的儲存必定不會命中另一工作者中的儲存的規則，除非它們是相同的儲存。其同時執行在相同調準時距中的一個工作者中的載入必定不會抵觸另一工作者中的儲存的規則。其在載入碰巧被首先處理並且儲存被稍後處理的情況中執行。若該儲存首先處理而載入於後，則會被載入正確性快取所捕捉。

如上所詳述，各工作者邏輯處理器具有一載入正確性快取。載入正確性快取識別所有自進入流至現行暫時性全域提交點已經儲存的位元組。特別是，其告知哪一個工作者爲該位元組的邏輯上最近的寫操作者。

寫回狀態機器工作於複製回復清單以將所有指示儲存合併入原始程式位址區位並且使得它們全域可見。複製回復清單代表非依程式順序的儲存的集合。因此所有表示在複製回復清單中的儲存必須變得基元性全域可見。外界必須完全看見這些儲存或是完全看不見。

若因爲諸如一執行唯一分支違反或一儲存違反或一載入違反之類的內部爭議而有一流退出，狀態將會在流退出之前回復至最近的全域提交點。這將會等待複製回復狀態機器來結束，如此確實是在違反之前的邏輯上最近的全域提交點。

若因爲諸如一失效窺探命中一全域推測載入之類的某種外部爭議而有一流退出，則在已經釋放給複製回復狀態機器的時距中有牽涉指令的可能性。除非得知不是這種情形，這必定會造成一立即退出而沒有全域提交複製回復狀態機器所現行從事的複製回復清單。

從資料快取的觀點，合併狀態機器像是另一資料快取。若合併狀態機器存取一快取線，這會窺探資料快取並且若該線於資料快取中爲髒則會迫使一寫回。

修改區塊可延伸到多重的中階快取以便具有少量同步

阻擋對原始程式位址區塊的所有存取（對軌跡資料的存取將會是唯讀而可行）。暫時性位元能夠對此使用。

當運行最佳化（緒）碼時，原始碼可能因最佳化碼或同步運行的非相關碼（或甚至是相同碼）所產生的儲存而改變。XMC Guardata 快取（XGC）409 使用以防範。此快取保持被存取處的所有分頁的（分頁間隔中的）位址，以產生最佳化碼以及將會在窺探失效命中的情況中使用的最佳化區域 ID。區域 ID 代表其聯集接觸該防範區域（快取線）的所有靜態碼線。圖 7 繪示根據一些實施例的 XGC 409 的使用。

在執行最佳化碼之前，該核心保證 XGC 中的所有入口存在並且不會被窺探或替換出。在那種情況中，允許執行最佳化碼。

若，在最佳化碼執行中的期間，另一邏輯核心改變任一原始分頁中的資料，XGC 將接受一窺探失效訊息（像是一致性域中的任何其它快取代理）並且將通知該核心之一必須放棄執行關聯於給定分頁的最佳化碼並且使得任何用於此分頁所保有的最佳化碼入口點失效。

當執行於緒執行模式時，各儲存對照 XMC 409 來查對以防範自我修改碼。若一儲存命中一熱碼區，將會觸發一違反。

在一些實施例中，緒產生器作出一些假設，在之後會被查對正確性。這是主要的性能最佳化。這類假設之一範例爲一呼叫返回配對。緒產生器可能假設一返回將會回至

其呼叫（絕大多數時間是正確的）。從而，緒產生器可將整個呼叫函數放入一個緒並且（在返回之後）允許跟隨的碼於其它緒中執行。由於跟隨的碼將會在執行返回之前開始執行，並且查詢堆疊，執行可能會錯誤（例，當該函數將一返回位址寫至該堆疊，重寫原始返回位址）。爲了防範這類情況，各緒能夠將斷言寫至斷言硬體區塊。若該些緒皆對該斷言意見一致，則滿足一斷言。爲了提交一時距必須滿足斷言。

III. 軟體

動態緒切換執行（DTSE）軟體使用硬體擷取的概析資訊以界定被稱爲「流」的碼的重要靜態子集。在一些實施例中，這軟體具有自己的工作記憶體空間。流中的原始碼在這工作記憶體中改製。工作記憶體中的碼複本能夠被軟體修改。原始碼被確切保持其原始形式，在記憶體中的其原始位置。

DTSE 軟體能夠於 DTSE 工作記憶體中將流分解爲能夠在多重邏輯處理器執行的多重緒。若邏輯處理器在缺少此動作的情況下未被完全利用，則會進行。這藉由硬體可能會作的五件事成爲可能。

在任何情況中，DTSE 軟體會將碼插入 DTSE 工作記憶體中的流以控制（也許是 SMT）硬體處理器的處理，在大量邏輯處理器中。

在一些實施例中，硬體應繼續概析包括 DTSE 軟體已

經處理的流的運行碼。DTSE 軟體回應於改變行爲以修改其先前處理的碼。因此軟體將會系統地（若緩慢）改善處理的碼。

界定流

當 DTSE 硬體在其清單頂端具有一新 IP 時，軟體使得該 IP 成爲一新流的概析根。軟體將會指揮硬體自概析根取得一概析。在一實施例中，硬體將使得該概析著手下次執行命中概析根 IP 以及在之後擴展大約 50,000 動態指令，在一個連續試行中。記憶體中的緩衝器充滿所有載入與儲存的位址、直接分支的方向、以及間接分支的目標。包括有返回。藉此，軟體能夠在靜態碼中的概析根開始並且通過靜態碼追蹤概析路徑。能夠找到每一分支的實際目標並且得知所有載入及儲存的目標位址。

將經由這概析路徑命中的所有靜態指令界定爲處於流中。將經由這概析路徑命中的每一控制流路徑界定爲處於流中。將未經由這概析路徑命中的每一控制流路徑界定爲離開流。

在一些實施例中，DTSE 軟體將引導硬體再次自相同的根取得概析。將並非已經在流中的新指令或新路徑加入該流。軟體將會在取得不將指令或路徑加入流的概析時停止請求更多的概析。

流維護

在流已經界定之後，可被監控及修改。包括在已經對其產生新碼（也許在多重緒中）之後。若修改流，典型地代表應該再生該碼（也許緒碼）。

老化無效流

在一些實施例中，各流保持一「指數老化」平均流長度，L。在一實施例中，L 初始爲 500,000。當執行流時，假設流中執行指令的數量爲 N。則算出：L = 0.9 *（L + N）。若 L 曾小於流的一設定值（假設 100,000），則刪除該流。那也意指這些指令適格再次成爲熱 IP's，除非它們是在一些別的流中。

合併流

在一些實施例中，當執行一流時，若在動態指令（例，25,000 個）的一組數量之前有一流退出，則其熱碼入口點被設定去拿取一概析而不是執行熱碼。下次命中此熱碼入口點時，將會對自那入口點的一些指令（例，50,000 個）採取概析。這增加對此流的概析集合。

任何新指令及新路徑會被加入流。在一些實施例中，流分析與碼產生藉集合中的新概析而再次完成。

拓撲分析

在一些實施例中，DTSE 軟體執行拓撲分析。此分析可由下列動作之一個或多個所組成。

基本區塊

DTSE 軟體將流的碼打斷爲基本區塊。在一些實施例中，只有在概析中已經被觀測的聯合保持爲聯合。故即使在具有明確目標的流中有一分支，並且此目標在流中，這聯合若從未被觀測到在概析中發生則將被忽略。

所有未被觀測到採取概析的控制流路徑（邊線）被標記爲「離開流」。這包括對於經觀察到爲始終採取的分支的「否則跳至（fall though）」（不採取分支）的方向。

在概析中的分支單調（monotonic）（包括非條件式分支）將不結束基本區塊除非該目標爲一結合（join）。呼叫及返回爲結束基本區塊。

在上述進行後，DTSE 軟體目前具有基本區塊的集合以及基本區塊之間的「邊線」的集合。

拓撲根

在一些實施例中，使用各概析以引導流的靜態碼的遍歷（traversal）。在各呼叫的這遍歷中，呼叫目標基本區塊識別符被推到一堆疊上並且在各返回時該推疊被彈出。

儘管整體碼流很可能具有平衡的呼叫與返回，該流爲出自多少有些隨機開始及終止點的動態執行的片段。沒有理由認爲在流中的呼叫與返回是平衡的。

遭遇的各基本區塊歸類爲在藉由堆疊頂端的基本區塊識別符所識別的程序中，若有的話。

出自概析根的碼，以某種手段，初始將不在任何程序中。很可能在概析稍後將再次遭遇這碼，在那裡將被識別爲在程序中。最可能是有不處於任何程序中的一些碼。

拓撲分析的質取決於拓撲分析所用的根。典型地，要得到好的拓撲分析，根應該在被界定處在流中的靜態碼的最外邊程序中，即，在「非在任何程序中」的碼中。由硬體所找到概析根可能不是。因此 DTSE 軟體界定用於拓撲分析的拓撲根。

在一些實施例中，非在任何程序中的基本區塊中，識別出一子集的碼（R），以致（自 R 中的任何指令開始，但僅使用該流的邊線，即，已經被觀測到進入至少一概析的邊線）會有到流中每一其它指令的一路徑。R 有可能爲空。若 R 爲空則界定拓撲根爲概析根。若 R 爲非空，則挑選 R 中數字最低的 IP 值做爲拓撲根。從此之後，任何提及的「根」意指拓撲根。

程序嵌入

傳統的程序嵌入用於消除呼叫與返回過載。DTSE 軟體持有關於碼行爲的資訊。程序中的碼依據對其呼叫的碼而有不同行爲。因此，在一些實施例中，DTSE 軟體對於這程序的每一不同靜態呼叫保持關於該碼之分離的資訊表格。

此碼的中間階段不是可執行的。當分析完成時，DTSE 軟體將產生可執行碼。在這中間階段中，沒有用於

嵌入的程序中的碼的複製。程序嵌入將多個名稱指派給程序中的碼並且將分別的資訊保持於各名稱中。

在一些實施例中，這是遞迴的。若外部程序（A）自3個不同地點呼叫程序（B），並且程序B自4個不同地點呼叫程序C，則對於程序C會有12種不同行爲。DTSE軟體將持有關於程序C中的碼的資訊的12個不同的表，對應於到此碼的12種不同呼叫路徑，以及對這碼的12種不同的名稱。

當DTSE軟體產生對於這流的可執行碼，這個碼的靜態複本很可能將會比12個還少很多。具有碼的相同位元的多重複本並不有趣，並且（在大部分情況中），呼叫及返回過載較小。然而，在一些實施例中，DTSE軟體持有對此碼的各呼叫路徑的分別的行爲資訊。DTSE所持有的行爲資訊的範例以指令相依最重要，及載入於儲存目標以及分支可能性。

在一些實施例中，DTSE軟體將認爲若有一呼交指令靜態處於原始碼中，出自該呼叫程序的返回將總是前往跟隨該呼叫的指令，除非被觀測到未在概析中發生。然而，在一些實施例中，會在執行對其正確予以查對。DTSE軟體所產生的碼將會來查對。

在一些實施例中，在DTSE對於原始碼中的呼叫指令所產生的最終可執行碼中，可能有用於該程式而將該架構返回位址推上該架構堆疊的一指令。注意這無法藉由所產生的碼中的呼叫指令而完成，因爲產生的碼位在完全不同

之處並且將會使錯誤值推上該堆疊。推上該堆疊的值極少用於熱碼。對於程式的資料空間將總是保持正確。若產生多重緒，並不使得緒的行爲有所差異。會在某時某處完成。

在一些實施例中，DTSE 軟體可能選擇將實體執行於線的特定緒的部分程序的實體拷貝予以放置，若該程序非常小。否則將不會有一實體複本在此並且將會有一些排序的一控制轉移指令，以前往該碼。這將會更加詳述於「碼產生」之段落下。

在一些實施例中，在 DTSE 對於原始碼中的返回指令所產生的最終可執行碼中，將有用於該程式的一指令將架構返回位址的架構堆疊中彈出。DTSE 軟體相信可能會是此返回的目標的該架構（非熱碼）返回目標 IP 將會被該碼得知。在一些情況中，這是熱碼中的一立即常數。在其它情況中，其儲存於 DTSE 記憶體中，也許是一堆疊結構。這不是該程式的資料空間的部分。自堆疊彈出的該值必須與 DTSE 軟體相信可能會是此返回目標的 IP 相比較。若這些值相異，則流退出。若產生多重緒，並不使得緒的行爲有所差異。會在某時某處完成。

在一些實施例中，DTSE 軟體將實體執行於線的特定緒的部分程序的實體拷貝予以放置，若該程序非常小。否則，將不會有一實體複本在此並且將會有一些排序的一控制轉移指令，以前往在此緒中返回目標之該熱碼。這將會更加詳述於「碼產生」之段落下。

往回邊線

在一些實施例中，DTSE 軟體會找到流的一最小往回邊線組。最小往回邊線組爲自一個基本區塊至另一個的邊線的集合，以至於若這些邊線被切斷，則將不會有封閉迴圈路徑。該組應在若任何邊線自該組中移除時仍會是一封閉迴圈路徑的判斷中爲最小的。在一些實施例中，有一特性，若在該組中的全部的往回邊線被裁切，該碼仍然爲完整連接。而可在基本區塊的整個集合中得自根而至每個指令。

各程序分別完成。因此呼叫邊線與往回邊線對此被忽略。

分別地，在一些實施例中可進行遞迴呼叫分析。其藉由巢套呼叫樹的探查而完成。自頂端開始，若有對已經在路徑上的巢套呼叫樹中的路徑上的任何程序的一呼叫，則有一遞迴呼叫。一遞迴呼叫爲一迴圈並且往回邊線爲界定出自該呼叫。故分別地，呼叫邊線能夠標記「往回邊線」。

在一些實施例中，演算起始於根並且追蹤從頭至尾每一個基本區塊的所有路徑。基本區塊的內部非是有形的。此外，已經被界定的往回邊線是無法橫跨的。若在自該根的任一線性路徑上（P），將面對一已經在 P 中的基本區塊（S），則這個在 S 終止的邊線將被界定爲一往回邊線。

界定分支再收斂點

在一些實施例中，有一些因爲採取成爲單調而未被預測的分支。若這個分支於執行中進入錯誤途徑，則爲一分支失誤預測。不僅如此，並且會離開該流。這些分支對所有用途被視爲絕對單調（即，一點條件分支也沒有），在處理流中的碼的過程中。

一間接分支將具有一已知目標清單。實質上，其爲一多重目標條件分支。DTSE 軟體可將其編碼爲比對與分支的連續字串，或具有一反彈表。在二編碼之一中，還有一個目標：離開該流。這實質上爲在末端的一單調分支。若走錯路，則離開流。於已知目標的多路分支具有相同於針對條件分支的一再收斂點，並且以相同方式得知。並且，當然，該非預測的、單調的最近再排序分支一點也不被作爲分支而控制。

呼叫及返回（如所述般）特別且不「分支」。返回是一再收斂點。程序 P 中的任何分支，其不具有以其它方式所界定的一再收斂點，具有「返回」作爲其再收斂點。P 可具有編碼於許多地方的返回。爲了成爲一再收斂點之故，所有返回的編碼範例採相同方式。對於程序的任何靜態實體，所有編碼的返回進入對該程序的靜態實體爲獨一無二的完全相同處。

藉由這全部，應該能夠找到對於全面分支的一再收斂點。在一些實施例中，只有對於基本區塊的入口點能夠是

一再收斂點。

對於一分支 B，可找到一再收斂點 R，以至於，遍及自 B 至 R 的所有控制流路徑，往回邊線遍歷的總數爲最小量。給定對於遍及所有自 B 至再收斂點的路徑的所有具有相同數量的往回邊線的分支 B 的再收斂點組，在自 B 至再收斂點的路徑的完整組具有最少指令的再收斂點典型地爲較佳。

在一些實施例中，在分析中保持二個參數：往回邊線限制及分支限制。皆初始爲 0。在一些實施例中，該處理爲通過尙未界定再收斂點的所有分支並且執行一個以上的下列動作。對各這樣的分支，B 在分支（B）開始隨著所有控制流路徑前進。若任何路徑離開該流，停止追蹤該路徑。若橫跨的不同往回邊線的數量超過往回邊線限制，則不再追蹤這個路徑並且使將要超過該限制的往回邊線不會越過。對各路徑，收集在路徑上的基本區塊的集合。找到所有這些集合的交點。若這個交點集合爲空，則此搜尋爲不成功。從該交點集合，挑選在自 B 至 R 的所有路徑上所有指令的總數爲最小值的該集合的成員（R）。

此刻，決定在自 B 至 R 的所有路徑中總共有多少「可見」往回邊線。若數量多於往回邊線限制，則否決 R。然後對於可見往回邊線的總數測試下一個可能具有較大量總指令的再收斂點。最後，發現滿足往回邊線限制的再收斂點，或是已無任何可能性。若找到一個，則確定在自 B 至 R 的所有路徑上尙未具有再收斂點的分支總數。若超過分

支限制，則否決 R。最後將找到往回邊線限制與分支限制皆滿足的 R，或是沒有可能性。一個好的 R 爲 B 的再收斂點。

在一些實施例中，一旦已經找到 B 的再收斂點，對於找到再收斂點的演算靜止，任何前進控制流遍歷通過 B 將直接跳躍至其再收斂點而不需察看該分支與其再收斂點之間的細節。任何後退控制流遍歷通過一再收斂點將直接跳躍至其匹配分支而不需察看該分支與其再收斂點之間的細節。實質上，控制流自分支收縮至其再收斂點降至一單點。

在一些實施例中，若找到再收縮點，接著該往回邊線限制及該分支限制皆重置，並且考慮尚未具有再收縮點的所有分支。若成功找到再收縮點，接著一些事件則變得不可見。此時，可能發現之前未成功的分支再收斂點，甚至在往回邊線限制與分支限制的更低值。

在一些實施例中，若沒發現再收斂點，試行下一個分支 B。當不具有再收斂點的所有分支已試行不成功時，則增量分支限制並且再次試行該些分支。在一些實施例中，若無潛在的再收斂點因分支限制而否決，則將分支限制重置至 0，增量往回邊線限制，並再次試行。

一般而言，能夠有其它分支（C）在自分支（B）至其再收斂點（R）的控制流路徑上尚未具有再收斂點，因爲該分支限制設定爲大於 0。對於各這樣的分支（C），C 取得指派到 B 所具有的一些再收斂點，即 R。該組分支（B

）以及所有這類分支（C）界定成爲一「分支群組」。這是一所有具有相同再收斂點的分支的群組。在一些實施例中，這是在自該些分支至再收斂點的整件事變得「不可見」之前處理。若未處理爲一群組，則任一分支一經取得指派一再收斂點，則找到群組中其它分支的再收斂點所不可或缺的所有路徑變得不可見，未提到的尙未有再收斂點的那些其他分支變得不可見。

在一些實施例中，所有分支具有界定的再收斂點。「一線性路徑中的往回邊線數量」意指相異邊線的數量。若相同往回邊線在一線性路徑中發生多次，仍算作只有一個往回邊線。若基本區塊（E）爲分支（B）的界定的再收斂點，則不使其失格成爲分支（D）的界定的再收斂點。

集體展開

在一些實施例中，執行集體展開。在集體展開中，創造一限量的碼的靜態複製以允許並行的特定形式的暴露。

在這些實施例中，整個流對各分支槽套層次被複製 N 次。N 的良好值可爲最終碼中所期望的軌跡數量，雖然其它數量可能具有一些優點。這個複製提供在多重（也許全部）軌跡中有相同碼而進行迴圈的相異重複的機會。不使任何迴圈相異重複進入相異軌跡。一些迴圈將皆由重複而分離，一些將在精細尺度指令藉由迴圈內的指令而分離。較一般地，迴圈將藉由指令基礎分離在一指令的兩式樣中。什麼要發生、將發生。其僅僅允許藉由重複的分離。

當事件坐落於這點時，僅有一迴圈體的一個靜態複本

。若僅有一個靜態複本，無法沒有動態複本而處在多重軌跡中，可能產生不良後果。爲了使這個碼處於多重軌跡中，用於相異控制流路徑（相異重複），應該有多重靜態複本。

巢套

一分支群組在自該群組至界定再收斂點的群組中的分支的路徑中具有至少一個可見往回邊線被界定爲一「迴圈」。特定分支群組爲「可見」或「不可見」是在再收斂點分析中界定。此外，不在自任一可見分支至其再收斂點的路徑上的任何往回邊線亦界定爲一「迴圈」。

一僅界定往回邊線的迴圈，係被界定爲具有自其往回邊線之起始，通過該往回邊線，回至該往回邊線之起始，而作爲「自其分支至它們的再收斂點」的路徑。

給定相異迴圈（A 及 B），若 B 的群組中的所有分支是在自 A 中的分支至 A 的界定再收斂點的路徑上，則 B 巢套於 A 中。一界定爲非處在一分支至其再收斂點的往回邊線的迴圈被界定爲不被巢套在任何其它迴圈內，但其它迴圈能夠巢套在它之中，且通常如此。

一僅界定成爲往回邊線的迴圈會與這個往回邊線相關。其它迴圈則相關於自迴圈的分支至迴圈再收斂點的路徑中的可見往回邊線。特定分支群組爲「可見」或「不可見」是在再收斂點分析中界定。

下列定理及輔助定理的一個或多個可應用於巢套的實

施例。

定理 1：若 B 巢套於 A 中，則 A 非巢套於 B 中。

假設 B 巢套於 A 中。則 B 中有一些分支，並且 B 中所有的分支是在自 A 至其再收斂點的路徑上。若 A 不含有分支，則根據定義，A 無法巢套於 B 中。若 A 中的一分支（X）是在自 B 中的分支至其再收斂點的路徑上，則若 X 是 B 的部分，就是它不可見於 B。若 X 爲 B 的部分，則 A 的全部是 B 的部分並且迴圈 A 與 B 並不相異。故 X 必須不可見於 B。這意指 A 必須在 B 進行之前已經具有界定的再收斂點，以使 A 的分支不可見於 B。因此 B 並非不可見於 A。B 中的所有分支是在自 A 至其再收斂點的路徑上並且是可見的。這使得 B 爲 A 的部分，故 A 與 B 並不相異。X 無法如假設般成立。

輔助定理 1：若分支 B2 是在自分支 B1 至其再收斂點的路徑上，則自 B2 至其再收斂點的整個路徑也會在自 B1 至其再收斂點的路徑上。

自 B1 至其再收斂點的路徑（R1）通向 B2。因此其跟隨著所有出自 B2 的路徑。若 B1 具有再收斂，則 B2 具有再收斂。若尚未到達對於 B2 所指定的「再收斂點」，則 R1 爲較佳點。再收斂點演算將會找到最佳點，故必須已經找到 R1。

定理 2：若迴圈 B 的一個分支是在自迴圈 A 中的一分支至其再收斂點的路徑上，則 B 巢套於 A 中。

假設 X 是在自 A 中的一分支至 A 的再收斂點（RA）

的路徑上的 B 中的一分支。根據輔助定理 1，自 X 至其再收斂點的路徑（RB）是在自 A 至 RA 的路徑上。迴圈 B 爲自 X 至 RB 的路徑上的所有分支的集合。它們全都在自 A 至 RA 的路徑上。

定理 3：若 B 巢套於 A 且 C 巢套於 B，則 C 巢套於 A。

假設 X 是帶有再收斂點 RC 的 C 中的一分支。則 X 是在自 B 中的分支 Y 至 B 的再收斂點（RB）的路徑。根據輔助定理 1，自 X 至 RC 的路徑是在自 Y 至 RB 的路徑上。B 中的分支 Y 是在自 A 中的分支 Z 至 A 的再收斂點（RA）的路徑上。根據輔助定理 1，自 Y 至 RB 的路徑是在自 Z 至 RA 的路徑上。

因此，自 X 至 RC 的路徑是在自 Z 至 RA 的路徑。故當然 X 是在自 Z 至 RA 的路徑上。這對所有 C 中的 X 皆成立。故 C 巢套於 A 中。

定理 4：一往回邊線會相關於一個且唯一 1 迴圈。

非在自一可見分支至其再收斂點的路徑上的往回邊線本身是一迴圈。若該往回邊線是在自一可見分支至其在收斂點的路徑上，則這個分支所屬的分支群組具有至少一個往回邊線，並且故而爲一迴圈。

假設有一往回邊線（E）相關於迴圈（L）。假設 M 爲一區別迴圈。若 L 或 M 是沒有分支的迴圈（即，僅是一單一往回邊線），則該定理爲真。故假設 L 與 M 皆具有分支。連續界定再收斂點。若 M 的再收斂點首先界定，

並且 E 在自 M 至其在收斂點的路徑上，則 E 會已經被隱藏。而之後不會可見於 L。若 L 的再收斂點首先界定，則 E 會隱藏並且之後不會可見於 M。

非定理 5：在一流中執行多於一次的所有碼是在一些迴圈中是不真實的。

在非在任何迴圈中的流中但是執行多次的碼的一實體是在一分支中終止的二個基本區塊。分支的一臂將該第一基本區塊作爲目標，分支的另一臂則將該第二基本區塊作爲目標。分支的再收斂點是到該第二基本區塊的入口點。第一基本區塊中的碼是在迴圈中但是在第二基本區塊中的碼並不在迴圈中，亦即，非在自該迴圈分支至其再收斂點的任何路徑上。

一「逆往回邊線」是一邊線相關於一迴圈分支群組，以至於自這個邊線前進，該迴圈分支群組的再收斂點在這個迴圈分支群組中的任何分支之間命中（並且可能從未命中這個迴圈分支群組中的任何分支）。若一往回邊線可見於一迴圈分支群組並且位在自那個迴圈分支群組的一分支至那個迴圈分支群組的再收斂點的路徑上，則該往回邊線是「相關於」那個迴圈分支群組。

注意，在一帶有退出迴圈的迴圈分支的典型迴圈中，通過該往回邊線的路徑最先命中該迴圈分支並且接著其再收斂點。若該往回邊線爲一逆往回邊線，通過這個往回邊線的路徑最先命中該再收斂點並且接著該迴圈分支。

定理 6：若不在任何迴圈中的一流中有執行多於一次

的一指令，則這個流含有一逆往回邊線。

假設 I 爲一在一流中執行多於一次的指令。假設 I 不在任何迴圈中。假設在該流中沒有逆往回邊線。

在自 I 返回 I 的流中必定有一些路徑（P）。在那個路徑中有至少一個往回邊線（E）。

假設有一分支 B 爲相關於 E 的一迴圈的部分。這意指 B 爲一分支群組的部分。E 可見於那個分支群組並且 E 在自那個群組中的一分支至其再收斂點的路徑上。

自 E 的前進是位在 P 上，除非有其它分支。若有其它分支（C），則 C 是位在自 B 至 B 的再收斂點的路徑上，因此 C 是在這個相同的分支群組中。C 在 P 中。因此，在 P 中有這個迴圈的一迴圈分支。若沒有 C，則 P 正被跟隨並且將抵達 I。若在 B 的再收斂點之前到達 I，則 I 在此迴圈中，與假設相違。故應該在到達 I 之前到達 B 的再收斂點。並且是在到達任何分支之前。故出自往回邊線的路徑在命中另一迴圈分支之前命中該再收斂點。

另一方面，假設有一迴圈分支（C）在 P 中。若再收斂點不在 P 中，則 P 全部在該迴圈中，特別是 I。故該再收斂點也在 P 中。故 C、E、及再收斂點（R）全都在路徑 P 上。序列必須進行到 E 然後 C 再來 R，因爲任何其它序列會給出一逆往回邊線。若在 P 上有多於一個的分支，諸如一能夠通到 P 上任意地方的分支（X）。但是至少一個迴圈分支必須在 E 與 R 之間。C 爲那個迴圈分支。

C 具有另一臂。應該有一自 C 的其它臂至 R 的路徑。

若自 C 通到 R 的所有路徑在 E 之前，則 E 不在自 C 至 R 的任何路徑。因此，整個自 C 至 R 的結構將不會可見於 B，並且 C 能夠不是這個迴圈的迴圈分支。因此出自 C 的一些路徑必須在 R 之前通過 E。但是這是不可能的。這個路徑必須在邊線 E 之前的某處加入 P。無論在哪，該處將是再收斂點（R）。結論是在 P 上的唯一可能的序列（從其它觀點來看，E 然後 C 然後 R）在事實上是沒有可能的。

在一些實施例中，依據上述一個或多個定理，迴圈可指派到一唯一的巢套層次。不具有其它迴圈巢套於內的迴圈處於巢套層次 0。含有該些迴圈的迴圈爲巢套層次 1。有最高巢套層次的一迴圈。這迴圈定義此流的巢套層次。要注意迴圈巢套僅處於程序內。其在各程序中自 0 重新開始。這因爲程序嵌入而相一致。該流的巢套層次是遍及該流中的所有程序的最大巢套層次。

由於各往回邊線屬於一個且唯一一個迴圈，往回邊線的巢套層次可定義成其所屬的迴圈的巢套層次。

在一些實施例中，DTSE 軟體將會複製整個流，爲一單元，N^U 次，其中 U 爲該流的迴圈巢套層次。N 爲呈現各迴圈巢套層次的展開之路的數量。

在一些實施例中，由於這是完全相同的碼的 N^U 精確複本，對於軟體沒有理由實際複製該碼。位元會完全相同。該碼概念上複製 N^U 次。

該流的靜態複本能夠以 U 數位號碼來命名。在一實施例中，該數位爲底數 N。最低階數位相關於巢套層次 0。

次一階數位相關於巢套層次 1。各階數位對應於一巢套層次。

在一些實施例中，對於各數位 D，在展開複本名稱中，DTSE 軟體使得帶有相關於 D 的巢套層次的每一往回邊線（在對 D 帶有值 0 的所有複本中）走到對 D 帶有值 1 的複本中的相同 IP，但所有其它數位相同。其使得帶有相關於 D 的巢套層次的每一往回邊線（在對 D 帶有值 1 的所有複本中）走到對 D 帶有值 2 的複本中的相同 ID，但所有其它數位相同。類推至複本 N-1。軟體使得帶有相關於 D 的巢套層次的每一往回邊線（在對 D 帶有值 N-1 的所有複本中）走到對 D 帶有 0 的複本中的相同 IP，但所有其它數位相同。

對此的實施例是現行展開靜態複本號碼以及當追蹤流時如何改變的一演算。這演算爲，若在向前方向中追蹤層次 L 的往回邊線，則第 L 數位模數 N 增量。若在向後方向中追蹤層次 L 的一往回邊線，則減量第 L 數位模數 N。這就是之前複合段落所說明的。在一些實施例中，DTSE 軟體不具有指標或任何代表這的東西。僅具有這單純的現行靜態複本號碼以及計數演算。

因此，在一些實施例中，DTSE 軟體已經藉由因數 N 展開所有迴圈。其立即集體進行而並未確實理解任何迴圈或對它們個別地查看。其僅確實得知的是：各往回邊線的巢套層次，及其最大值，該流的巢套層次。

在這些實施例中，由於沒有目標 IP 改變，故沒有改

變碼中的任一位元。所改變的是在相同 IP 的指令的各靜態實體能夠具有不同的相依性。各靜態實體取決於不同的其它指令以及不同的其它指令取決於它。對於各指令，由其 IP 所界定，期望有記錄其個別對每一個靜態實體的相依性的能力。當追蹤任何控制路徑，一展開複本計數器將適當地改變狀態以常時立即告知正在對哪個指令的展開複本進行查看。

分支再收斂點

在一些實施例中，若（在控制流圖表遍歷中）命中一分支（B），其爲迴圈（L）的一部分，則 B 所屬的分支群組之一識別符被推到一堆疊上。若（在控制流圖表遍歷中）命中分支群組已經位在堆疊頂端的一分支，則不作任何動作。若在控制流圖表遍歷中對位在（展開前所界定的）堆疊頂端的分支命中一再收斂點，X，則進到這個展開巢套層次的版本 0，並且彈出堆疊。這說明 X 的版本 0 將會是此展開迴圈的實際的再收斂點。

在一些實施例中，有一例外。若所追蹤的 L 的最近往回邊線爲一逆往回邊線並且命中（在展開前所界定的）L 的再收斂點，X，並且 L 位在堆疊頂端，則該堆疊被彈出，但是應維持相同的展開版本而不是進到版本 0。在這個情況中，X 的這個展開巢套層次的版本 0 被界定爲 L 的再收斂點。

在退出一迴圈（L）時，總是進到 L 的巢套層次的版

本 0（除非當 L 具有一逆往回邊線時）。

上述說明如何跟隨控制流圖表前進的實施例。當在一些實施例中出現時，可能比起前進更需要跟隨控制流圖表後退。在一些實施例中，那與巢套程序相同。

回至首次命中的 L 的再收斂點。困難在於這可能是多迴圈且同時是非迴圈的分支群組的再收斂點。問題在於哪個結構是正要退入的？這確實能夠有許多去這個點的路徑。若退入一迴圈，其應該現行點之下的巢套層次 1。可能仍有許多迴圈在這個巢套層次，以及有非迴圈分支群組。可能作出對哪一個正跟隨路徑的挑選。若挑選正要退入的一迴圈（L），有 N 路徑以跟隨入 N 展開複本。在一些實施例中，挑選這些之中的一個。現在得知正要退入的碼的靜態複本。所可能要查看的是在對應分支群組中的一分支。那資訊會被推到堆疊上。

在一些實施例中，若不在現行巢套層次的展開複本 0 中，則退入這個迴圈的一往回邊線。因此，當取得往回邊線的最近機會來臨時，則得知該路徑。直到此則有所有的可能性。若在現行巢套層次的展開複本 0 中，則不取得任何往回邊線的額外選擇，且在一些實施例中可能使得備份整個迴圈。若該迴圈被整個備份，彈出該堆疊。

在一些實施例中，每一時間對這個迴圈的一往回邊線採取減量在這個巢套層次模數 N 的複本號碼。

迴圈典型地是在其巢套層次的靜態複本 0 進入，並且總是退出至其巢套層次的靜態複本 0。

記住，內部有作業的是正在分析碼而非是正在執行碼的軟體。在大部分的實施例中，執行不具有這樣的堆疊。將會產生碼以恰好進到正確位置。對於產生該碼以進到所有正確位置的軟體而言，必須得知其本身如何追蹤流。圖8至11繪示一些這些作業的範例。圖8顯示三個基本區塊帶有二個往回邊線的一範例。此形成二個層次的巢套簡單迴圈。到C的入口是B中分支的再收斂點。自C的出口的目標是C中分支的再收斂點。圖9顯示整個流已經被複製。其一部分顯示於此。目前有這個巢套迴圈的4複本——複本00、複本01、複本10、及複本11。到Cx的入口是Bx中分支的再收斂點。自Cx的出口的目標是Cx中分支的再收斂點。各x有不同。圖10顯示已經使用上述討論的一或多個作業而修改的到再收斂點的往回邊線以及邊線。到C00的入口目前是迴圈B00-B01的再收斂點。到C10的入口點目前是迴圈B10-B11的再收斂點。外圍迴圈，靜態複本00及10皆進到共同的再收斂點。有一共同收斂點也是C01與C11的目標。這因爲C01與C11爲死碼故較無興趣。沒有達到這碼的方式。事實上，出自這片段的碼的出口總是在出自C00或C10的靜態複本C00中。在圖11中，已經移除死碼及死路徑以更清楚顯示其如何工作。注意僅有一個活入口到靜態複本00中的這個碼，以及自靜態複本00中的這個碼僅有一個活出口。在一些實施例中，DTSE軟體不會特地「移除」任何碼。只有該碼的一個複本。沒有可移除的東西。軟體的確理解基本區塊A與

C 只需要附屬於二個名稱：00 與 10 的相依資訊，而非屬於 4 名稱。基本區塊 B 需要附屬於四個名稱的相依資訊。

數字較大的 N 使工作量增加以準備該碼，但可能也潛在地隨著較少動態複製而增加並行性。在一些實施例中，DTSE 軟體可能增加 N 以使進行較佳工作，或是減少 N 以生產帶有較少工作的碼。一般而言，匹配軌跡的最終數字的 N 將得到大部分的並行性與一合理的工作量。一般而言，比這個更大的 N 將產生一少量較佳結果與一更大量的工作。

迴圈展開對一些重複提供在一個軌跡中執行指令（I）的可能性，當對一不同重複的相同指令（I）的一不同靜態版本被同時執行在一不同軌跡時。在此強調「指令」，因爲軌跡分離藉由指令基礎在一指令上完成。當在這個迴圈中恰好次於 I 的指令（J）可能被徹底不同地操作時，指令 I 可能以此方式操作。當對軌跡 1 中的全部重複執行在這個迴圈中恰好次於 I 與 J 的指令（K）時，可能對軌跡 0 中的全部重複執行指令 J。

迴圈展開，允許出自相同迴圈之不同重複的指令被執行在不同軌跡中，是一種有用的工具。在許多碼中揭開顯著的並行性。另一方面，迴圈展開在許多碼中絲毫沒有揭開並行性。這是 DTSE 可使用的工具中的唯一一個。

再者，對於 DTSE 軟體內的分析，典型地沒有理由複製任何碼用於展開，因爲位元可能完全相同。展開產生對於碼的多個名稱。各名稱具有其自己的屬性表。各名稱能

夠具有不同的行爲。這可能揭開並行性。甚至之後將產生的可執行碼不會具有許多複本，即使如此，在分析期間，這個碼有許多名稱。

線性靜態複製

在一些實施例中，整個流已經被複數許多次用於集體展開。除此之外，在一些實施例中，整個流視需要被複製多次。這些複本被命名爲 S0、S1、S2、...。

流中的各分支（B）被複製於各靜態複本 S0、S1、S2、...中。B 的各複本爲一般性分支（B）的實體。相似地，B 具有目前已複製於 S0、S1、S2、...中的再收斂點。所有的複本爲一般性分支（B）的一般性再收斂點的實體。複製的往回邊線全部標記爲往回邊線。

在一些實施例中，沒有複製任何碼。在那些實施例中，碼中的每樣東西取得多名稱的再另一層次。每一名稱取得一位置以儲存資訊。

在一些實施例中，流的全部「S」複本中的所有邊線使其目標改變至正確的一般性基本區塊，但不指派於一特定的「S」複本。所有往回邊線使目標明確改變至 S0 複本。

在一些實施例中，流 S0、S1、S2、...的複本依數值次序一個接一個完成。對於 Sk，在流複本 Sk 中帶有原址的每一邊線（E）（非爲一往回邊線）對其目標指派該特定複本爲最低的「S」號複本，以不與任何其它邊線共享一

目標。

最終，將沒有任何不爲往回邊線的邊線與其它任何邊線共享一目標基本區塊。往回邊線當然將頻繁地與其它（也許是很多其它）往回邊線共享一目標基本區塊。

在集體展開的情況中，在一些實施例中退出迴圈的邊線的目標「S」實體如下述般藉由走到迴圈再收斂點而修改。

在一些實施例中，若（在控制流圖表遍歷中）命中是爲迴圈（L）的一部分的一分支（B），B 所屬的分支群組的一識別符被與現行「S」實體號碼被推入堆疊。在一些實施例中，若（在控制流圖表遍歷中）命中分支群組已經在堆疊的頂端的一分支，則不作任何動作。在一些實施例中，若命中位在堆疊頂端的迴圈的一般性再收斂點的實體，在控制流圖表遍歷中，則彈出該堆疊並且實際走到自堆疊彈出的「S」實體號碼。

這說明在迴圈中，迴圈的各重複起始於「S」實體號碼 0，但當退出此迴圈時，走到進入此迴圈的「S」實體。

需注意能夠使用相同於集體展開所使用的堆疊。若使用相同堆疊，添加一區域至各堆疊成分，以用於該「S」實體。

再者，這些是正分析碼而非執行碼的軟體內部的作業。執行不具有這樣的堆疊。碼將被產生到恰好全部到正確位置。對於產生碼到全部正確位置的軟體，必須得知本身

如何追蹤流。

這將是第一流複本 Sx，其自複本 S0 無法到達。不需要這個以及所有較高編號的複本。除此之外，各殘留靜態複本 S1、S2、...典型地具有許多自 S0 無法到達的死碼。無法自此處而到達的東西將不產生送出的可執行碼。

相依性分析

多結果指令

在一些實施例中已經討論過原始呼叫指令可能已經被一推入所取代並且原始返回可能已經被一彈出所取代並且比對。

一般而言，在分析中不期望有多結果指令。在一些實施例中，會將這些分離成多指令。在許多（但當然並非全部）的情況中，這些或相似指令在碼產生時可能被重組。

推入及彈出是明顯的範例。推入是一儲存以及一減量堆疊指標。彈出是一載入以及一增量堆疊指標。時常會期望分離堆疊指標修改以及記憶體作業。有許多具有多結果之其它指令能夠被分離。在一些實施例中，這些指令被分離。

將這些分離的共同理由在於（非常可能）所有緒會需要追蹤堆疊指標改變，但應該沒有必要複製推入每一緒中的資料的計算。

不變值

硬體支援及機構

在一些實施例中，DTSE 硬體具有一些「斷言暫存器」有效於軟體。各「斷言暫存器」能夠至少保持二個值：實際值以及斷言值，並且各值帶有一有效位元。在一些實施例中，斷言暫存器爲對於所有核心及硬體 SMT 緒的一全域資源。

在一些實施例中，DTSE 軟體能夠在任何時間寫操作任何斷言暫存器的實際值部分或斷言值部分，始於任何核心的任何硬體 SMT 緒。

在一些實施例中，爲了全域提交一寫操作至一給定斷言暫存器的斷言值，目標斷言暫存器的實際值部分必須有效且二值必須匹配。若實際值並非有效或是值並不匹配，則硬體將引起一髒流退出，並且狀態會回復至最近全域提交狀態。

斷言暫存器提供碼運行在一個邏輯處理器（A）的能力，在一個核心使用一非實際藉由這個邏輯處理器或核心計算出的值的過程中。在一些核心的一些邏輯處理器（B）中，那個值必須邏輯上較早算出，但不必要實體上較早，並且寫入至斷言暫存器的實際值部分。運行在 A 中的碼能夠假定任意值並且將其寫入至相同斷言暫存器的斷言值。跟隨斷言值的該寫操作的碼能夠肯定，寫入至斷言值的值的確實匹配寫至在該寫至斷言值的邏輯位置的實際值的值，無論這個碼被放置在何處。

這是有用的，當 DTSE 軟體對於得知一值而不需進行

該值的所有運算，並且這個值用於多件事情具有高可能性，但非必然時。提供在多核心中的多邏輯處理器中使用這個值的可能性但是僅在一個核心的一個邏輯處理器中作正確運算。在 DTSE 軟體對於該值爲正確的事件中，基本上於斷言運作沒有花費。若 DTSE 軟體對於該值並非正確，則沒有正確性爭議，但是可能對於導致的流退出有大的性能花費。

堆疊指標

堆疊指標及基指標一般被頻繁使用。不大可能執行許多有用的碼卻不使用堆疊指標及基指標中的碼。因此，典型地，每一 DTSE 緒中的碼會使用這些暫存器的大部分的值。同時一般，例如，堆疊指標的實際值會取決於對於該堆疊指標的改變的一長相依性鏈。在一些實施例中，DTSE 軟體能夠打斷這個長相依性鏈，藉由於斷言暫存器的實際值部分插入一寫操作，接著藉由對斷言暫存器的斷言值作假定值的寫操作。然後有不直接相關於實際值的寫操作、或任何進行其者之值。

對於原始碼中的程序呼叫與返回，DTSE 軟體通常會假定恰在返回之後的堆疊指標與基指標的值與恰在呼叫之前的值相同。

恰在呼叫（原始指令）之前，在一些實施例中可能有一虛擬指令插入。這是一種不會產生碼的指令，但是具有如指令般的表格。該虛擬被標記爲堆疊指標與基指標的消

費者。

在自該程序的返回之後，指令被插入以複製堆疊指標與基指標至 2 斷言暫存器的實際值部分。這些插入的指令被標記爲這些值的消費者。

恰好在此之後，在一些實施例中，指令被插入以複製堆疊指標與基指標至這些斷言暫存器的斷言值部分。這些插入的指令被標記爲不會消費這些值，而是產生這些值。這些指令被標記爲直接相關於該虛擬。

相似地，對於許多非明顯進行不平衡堆疊改變的迴圈而言，假設堆疊指標與基指標的值會在各重複的起始處相同。爲消費者的虛擬，在一些實施例中，插入在對於迴圈的初始入口。於實際值的複本被插入並識別爲消費者，隨著於斷言值的複本被識別爲生產者。於斷言值的這些複本會直接依據該虛擬而製成。

許多其它使用能夠由此製成。注意於使用一斷言，不變的值並非必須。僅必須一多步驟評估能夠被很可能正確的一非常更短評估所取代。

斷言比對失敗由硬體作回報。若一斷言明顯失敗，在一些實施例中，DTSE 軟體會移除違法斷言暫存器使用並且對碼再加工而沒有失敗斷言。

注意，產生與這個一致而錯誤的碼是相當可能的。緒可能隨著一些但非全部的改變而結束於程序中的堆疊指標。因此能夠對於在程序末端的堆疊指標假設錯誤值。這不是一正確性問題。斷言會抓到它，但是斷言會總是或頻繁

地失敗。若緒並非將要具有程序所有的堆疊指標改變，則會想要使它一點也沒有。這並間接強制的。

具有於實際值得寫操作的緒會具有於堆疊指標的所有改變。這不是一常見問題。在一些實施例中，若在執行中回報有斷言失敗，則移除該斷言。

在一些實施例中，DTSE 軟體能夠明確查對於一緒中的一假設不變值的一些但非全部改變。若偵測到這個問題的情況，則移除該斷言。或者該值能夠儲存在該虛擬的位置並再載入在該斷言值的寫操作的位置。

控制相依性

在一些實施例中，使用各概析通過完整複製碼去追蹤一線性路徑。該概析界定各分支或跳躍的一般性目標並且在完整複製碼中的可行性路徑界定作目標的特定實體。因此這個追蹤將通過指令的特定實體。概析是一線性清單但是以間接方式通過完整複製靜態碼。一般而言。會多次命中相同指令實體。個別對於各分支的各靜態實體而言，取得各離開邊線花多少時間的記錄。

若出自一分支的實體的邊線尚未被視爲受到概析，則這個邊線正離開流。這可能使一些碼成爲無法到達。一分支的單調實體標記爲一「唯執行」分支。這些中的許多識別在先。一般性分支可能爲單調。在這個情況中，這個一般性分支的所有實體爲「唯執行」分支。於是，即使該一般性分支非是單調，這個分支的某些靜態實體可能單調。這些實體也是「唯執行」分支。

其它指令分支從不取決於「唯執行分支」。具體分支實體是或不是「唯執行」。

在一些實施例中，對於一般性分支（B）的各非唯執行實體，前溯所有路徑，停在 B 的一般性再收斂點的任何實體。在這個路徑上的所有指令實體標記具有對 B 的這個實體的一直接依賴。在一些實施例中，對所有一般性分支（B）完成。

可能有一分支具有「離開流」作爲一離開邊線，但是具有多於一個的其它邊線。對於間接分支而言是典型的。概析已識別該間接分支的一些可能目標，但典型地是假設有未識別的目標。若該間接分支走到一未在概析中識別的目標，即爲「離開流」。

在這些情況中，DTSE 軟體將這個打斷成爲一分支於已知目標以及一「離開流」或否的雙向分支。「離開流」或否分支是一種典型的單調「唯執行」分支。

直接相依性

在一些實施例中，各指令實體的直接控制相依性已經被記錄。

對於各指令實體，識別它的「暫存器」輸入。這包括所有需要執行該指令的暫存器值。這可能包括狀態暫存器、條件碼、以及內隱暫存器值。

在一些實施例中，做出對所有可能路徑自各指令實體的追溯，以找到需要的「暫存器」值的所有可能源頭。源

頭是一具體指令實體，非是一般性指令。具體指令實體自具體指令實體取得值。一單一需求值於一實體能夠有多源頭。

概析是分支目標以及載入位址與大小以及儲存位址與大小的一線性序列。DTSE 軟體應具有至少一個概析以進行相依性分析。數個概析可能爲可行的。

在一些實施例中，使用各概析通過完整複製碼去追蹤一線性路徑。該概析界定各分支或跳躍的一般性目標並且在完整複製碼中的可行性路徑界定作目標的特定實體。因此這個追蹤將爲通過指令的特定實體。概析是一線性清單但是以間接方式通過完整複製靜態碼。一般而言。會多次命中相同指令實體。

載入是自記憶體頻繁載入數個位元組。原則上，各位元組是個別相依性問題。實行上，當然地，能夠最佳化。在一些實施例中，對於各載入的各位元組，自概析中的載入以相反順序回顧以於這個位元組找到最近的先前儲存。存在載入指令的相同實體以及儲存的精確實體。在一些實施例中，這個儲存實體被記錄爲在這個載入實體中的一直接相依性。一載入實體可能直接取決於許多儲存實體，即使對於相同位元組而言。

超級鏈

沒有其它指令實體所直接取決於的各指令實體是爲「超級鏈產生器」。

超級鏈是含有超級鏈產生器的靜態指令實體組（在相依性下）的遞移閉包。亦即，引起超級鏈作爲含有超級鏈產生器的組。在一些實施例中，超級鏈中的任何指令實件（instance）係相依於是否有任何指令實踐被加入至該集合中。在一些實施例中，是連續遞迴直至該超級鏈含有在超級鏈所取決中的任何指令實體的每一指令實體。

在全部超級鏈已經形成自識別的超級鏈產生器之後，可能餘留一些不在任何超級鏈中的指令實體。在一些實施例中，任何不在任何超級鏈中的指令實體被挑出並指派成爲一超級鏈產生器及所形成的它的超級鏈。若仍餘留有不在任何超級鏈中的指令實體，挑出任何這類指令實體作爲一超級鏈產生器。這會持續直至每一指令實體是在至少一個超級鏈中。

注意許多指令實體將會在多數個（甚至很多個）超級鏈中。

在一些實施例中，該超級鏈組爲相依性分析的終產物。

軌跡形成

基本軌跡分離

在一些實施例中，若期望 N 個軌跡，則在相同時間分離 N 個軌跡。

初始種子產生

在一些實施例中，會找到最長的超級鏈（即爲「主幹」）。

對於各軌跡，在一些實施例中，會找到具有最多不在主幹中且不在任何其它軌跡中的指令的超級鏈。即爲這個軌跡的初始種子。

在一些實施例中，對軌跡組重複一次或二次。對於各軌跡，在一些實施例中，找到具有最多不在任何其它軌跡中的指令的超級鏈。即爲對於這個軌跡的下個重複，並且取代之前已有的種子。對於這個精煉，若確實看起來像是最有特色的選擇，則可能或可能不是允許主幹成爲種子的好主意。

典型地，這僅是「播種」軌跡的開始，並非結束。

軌跡栽培

在一些實施例中，挑出估計爲動態最短的軌跡（T）。然後將超級鏈放置在這個軌跡中。

在一些實施例中，會藉由尚未在任何軌跡中的動態指令的估計數而依序自最小到最大來複審超級鏈。

在一些實施例中，對於各超級鏈，對照於將其放置在任何其它軌跡中，若會造成一半以下的複製將其放置在軌跡 T 中，則如此放置，並且追溯至軌跡栽培的起點。否則跳過這個超級鏈並且嘗試下個超級鏈。

若已經到達沒有放置在軌跡 T 中的超級鏈的清單的結尾，則軌跡 T 需要一新種子。

新種子

在一些實施例中，自 T 以外的所有軌跡移除所有「成長的」超級鏈，在這些軌跡中留下所有「種子」。暫時地，軌跡 T 保持它「成長的」超級鏈。

在一些實施例中，自未安置的超級鏈的現行池，會找到具有非在 T 之外的任何軌跡中的指令的（動態評估的）最大數的超級鏈。這個超級鏈即爲軌跡 T 中的追加種子。

於是自軌跡 T 中移除所有「成長的」超級鏈。「成長的」超級鏈已經自所有其它軌跡移除。所有軌跡此刻僅含有其種子。在各軌跡中能夠有多個，甚至許多個種子。

從此處可執行軌跡栽培。

得到好種子有助優質軌跡分離。最長的超級鏈很可能是具有整組「主幹」指令的一個，其將非常可能在所有軌跡中結束。非常可能未界定一區別組指令。因此不是在一開始選擇成爲種子。

在一些實施例中，反而，期待超級鏈帶有盡可能多的相異於「主幹」的指令。具有好機會於區別。各相繼軌跡取得盡可能相異於「主幹」的種子以同時具有最好機會於區別，以及盡可能相異於現存的軌跡。

在一些實施例中，這會再次作重複。若有對於各軌跡的一些事，若可能則作出使得各軌跡更爲區別的嘗試。對各軌跡中的種子的選擇重新考慮僅可能相異於其它軌跡。

此後，可能有一雙岔途徑。

「栽培」預期是極度增長的。其僅僅使已經存在於軌跡中增加一些且只有其相當清楚其已經屬於此軌跡。「栽培」不會造成大跳躍。

在一些實施例中，當一停止出現明顯增長的栽培，則做出跳躍至活動的一新中心。爲此，增加軌跡中種子的收集。

大跳躍藉由增加一種子而完成。栽培填入顯然伴隨種子之物。一些流將具有非常好的連貫性。來自初始種子的增長的栽培可能工作相當順利。一些流會具有多個階段。各階段具有一種子。於是軌跡會遞增地替代得很好。

在一些實施例中，爲了找到軌跡 T 的新種子，所有其它的軌跡除了種子之外爲空。在新種子上可能具有的期望偏差爲何。然而，我們想要保留在軌跡 T 中所擁有的每樣東西。這是已經自然相關於 T 的東西。所想要的是找到不同東西進入 T。它不會幫助我們使得無論以何種方式所取得的東西成爲一種子。我們需要一些不會藉由栽培而取得的東西以作爲種子加入。

當返回至栽培，在一些實施例中，此過程乾淨地開始。栽培能夠對於種子中差異採實質相異方向並且這些種子可能是可最佳化的。

在一些實施例中，執行栽培一段時間作爲只是找尋種子所需要之物的機制。在流具有不同階段的情況中，可能需要所有不同階段中的種子。但是不知道階段或是需要多少種子。在一實施例中，這即是找出的方式。由於「試用

」栽培僅是一種發現種子所需的方式，只是投機方式。當有一整組需要的種子時，則做出一高品質「栽培」以填入在各軌跡中進行之物。

未加工軌跡碼

在一些實施例中，對於各軌跡而言，完整複製流爲開始點。自此之後，來自這個軌跡的碼的每一指令實體，對未在任何指派至此軌跡的超級鏈中的予以刪除。這即是這個軌跡的未加工碼。

一旦界定對於該軌跡的未加工碼，對於超級鏈沒有進一步作用。超級鏈僅存在用於決定哪個指令實體能夠自各軌跡的碼刪除。

在這點，所有軌跡含有所有完整複製基本區塊。實際上，僅有一般性基本區塊並且它具有許多名稱。對於它的各名稱，具有相異子集的指令。對於各名稱，其具有走到不同名稱的其他一般性基本區塊的離開邊線。一些離開邊線爲往回邊線。一般而言，在一些或甚至全部的名稱之下的許多基本區塊會不包含指令。

一基本區塊的各名稱具有其自身的離開邊線。甚至空間基本區塊實體具有離開邊線。可能或可能不是在一確定名稱的基本區塊中的分支及跳躍不會對於基本區塊正確地支援那名稱的離開邊線。有不包含跳躍或分支指令的基本區塊的實體（名稱），仍然有對於這個基本區塊之這個實體的離開邊線。未出示的分支與跳躍仍然具有原始碼目標

IP。依然有待修正。目標 IP 將必須改變以支援離開邊線，但尚未完成。並且對於基本區塊的許多實體，甚至將必須在結尾插入一控制轉移指令（跳躍）以支援離開邊線。

所有的軌跡具有確切相同的控制流結構以及確切相同的基本區塊實體，在這點。這全是相同東西，僅對於各軌跡帶有相異指令刪除。然而，對於軌跡的刪除能夠爲大，自整體結構撤除所有指令。舉例而言，在一迴圈中的所有指令可能已經自軌跡整體消失。

時距標誌

時距標誌指令是特有指令，在一些實施例中，於一 DTSE 暫存器的一儲存，也指出哪些其它的軌跡在碼中的相同地方也具有這個時距標誌。將會在之後補上。在產生可執行碼之前將不會知道。

在一些實施例中，任何往回邊線，非爲一逆往回邊線，以展開複本號碼數位的層次的展開複本 0 爲目標，取得插在往回邊線上的一時距標誌。這是僅含有時距標誌的一新的基本區塊。改變往回邊線而實際以這個新的基本區塊爲目標。這個新的基本區塊具有唯一一個，前往該往回邊線之先前目標的無條件離開邊線。

在一些實施例中，自這些時距標誌的邊線的所有目標取得僅在接連處之前插入的時距標誌。這個新的時距標誌不在出自該在往回邊線之時距標誌的路徑上。其在進到這接連處的所有其它路徑上。這個時距標誌同時是一新的基

本區塊，其僅含有時距標誌以及僅有一個進入該接連處的無條件離開邊線。

在一些實施例中，每一分支具有一逆往回邊線，這個分支的再收斂點取得加入作爲基本區塊中的第一指令的一時距標誌。

所有時距標誌將會匹配遍及所有軌跡，因爲所有軌跡具有相同基本區塊以及相同邊線。在可執行碼產生中，一些時距標誌將會自一些軌跡消失。可能必需保持軌跡的時距標誌匹配遍及軌跡，如此將會在它們其中一些消失時得知。

可執行碼產生

產生的可執行碼不具有靜態複本名稱或代表在 DTSE 軟體內部使用的資訊表格。在一些實施例中，標準 X86 指令相繼地執行，依位址次序，除非執行一分支或跳躍至不同位址。

這碼爲一「池」。它不屬於任何特定軌跡，或是另外任何東西。若該碼的一部分具有正確指令序列，任何軌跡能夠在軌跡中的任何地方使用它。若需求的碼已存在「池」中，沒有需要再次產生相同碼的另一複本。

當然，有一旦執行在一些碼中開始的爭議，碼自身決定將會執行的所有將來碼。假設有相同碼（C）匹配二個不同使用（U1 及 U2）的需求的指令序列，但在完成 C 的執行之後，U1 需要執行指令序列 X，而 U2 需要執行指令

序列 Y，並且 X 與 Y 不相同。這有可能是一問題。

對於 DTSE 碼產生，這問題有至少二個解。

在一些實施例中，第一個解是於 DTSE 軟體而產生碼的靜態複本的方式將使以下情形成爲頻繁：對於像是 C 的相同碼序列爲要求一段時間的像是 U1、U2 的不同使用，將事實上在其之後而永遠需要該相同碼序列。

在一些實施例中，第二個解是碼的一區段，諸如 C，匹配多個使用，諸如 U1 與 U2，能夠製成一 DTSE 子常式，U1 與 U2 在該子常式內使用相同碼（C），但 U1 與 U2 在自這個子常式返回後能夠爲不同。此外，創造該碼的靜態碼複本的碼分析軟體的方式通常明顯且容易使得這樣的子常式形成。這些子常式不被原始程式所得知。

建築區塊

該碼已經建造以自然地落入吊床（hammock）。吊床是成爲一 DTSE 子常式的固有候選。

DTSE 子常式並非原始程式所知的程序。注意，DTSE 子常式的返回位址並非正常地放上架構堆疊。除了它對於該程式不是正確之外，所有執行核心將分享相同架構堆疊，然而，一般而言它們執行不同版本的吊床且需要不同返回位址。

可能值得使用呼叫與返回址令以進到 DTSE 子常式與自該 DTSE 子常式返回，因爲硬體具有特定結構以非常精確地分支預測返回。在一些實施例中，堆疊指標在呼叫之

前改變成指向一 DTSE 私用堆疊，以及在執行碼之前改變回至程式堆疊指標。接著改變回至私用堆疊指標以返回。私用堆疊指標值必須儲存在一致定址但各邏輯處理器爲不同的一區位中。舉例而言，一般暫存器是這樣的儲存器。但是它們被用來執行程式。DTSE 硬體能夠提供一致定址的暫存器，但對於邏輯處理器特定儲存器予以存取。

如所提到的，頻繁不必要去製造一子常式，因爲會分享一碼序列的使用將（事實上）老是自這個點執行相同碼。若它的使用者對「老是」來自這個點的碼取得一致意見，則一可分享的碼序列將不會製造一子常式。

若一吊床版本的所有使用者在吊床之後進到相同碼，典型地沒有需要在這個點返回。共同碼只要對於所有使用者是相同的就能夠延伸。當使用者不再對要執行的碼取得一致意見時需要返回。

只有期望執行夠長以合理分攤呼叫與返回成本，吊床才會製造一子常式。若非真，則不製造子常式。

線內程序

程序已被「線內」化，產生它們的「複本」。這是遞迴的，所以僅需少量呼叫層級以及少量呼叫行，就能夠有大量的「複本」。另一方面，程序爲 DTSE 子常式之一良好候選。可能地，在大部分一般情況中的程序的許多「複本」，結果全都是相同的（除了針對不同軌跡的不同指令子集）。或者，它們可能成爲少數實際上相異的版本（除

了針對不同軌跡的不同指令子集）。因此程序成爲一個或僅少數的 DTSE 子常式（除了針對不同軌跡的不同指令子集）。

集體迴圈展開（unrolling）

在一些實施例中，一迴圈總是進入這個迴圈的展開複本 0。迴圈定義爲具有一單一出口點，這個迴圈的展開複本 0 中的迴圈分支群組的一般性共同再收斂點。這使得它成爲吊床。因此迴圈能夠總是成爲一子常式。

機會性子常式

分支樹的部分可能呈現爲在樹中反覆的一吊床。關於這的一不重要例子是分支的樹，具有線性靜態複製有效地解譯至許多線性碼片段。這些線性碼片段的一些暫時含有相同碼序列。線性碼序列能夠總是爲一子常式。

碼組合

在一些實施例中，對於各軌跡，拓撲根是開始點並且所有可到達的碼與所有可到達的邊線自此遍佈。碼在橫跨時產生。如何自特定基本區塊實體進到特定基本區塊實體在先前說明過了。

特定軌跡中的基本區塊實體可能不具有指令。則不產生碼。然而，可能有多個出自這個基本區塊實體的離開邊線應留意。

若在軌跡中的基本區塊的實體具有多個離開邊線，但至選擇該離開邊線的分支或間接跳躍自這個軌跡中的這個實體刪除，則這個軌跡在該分支的這個（刪除的）實體與它的再收斂點之間將不含有任何指令。在一些實施例中，該遍歷應不跟隨這個軌跡中的基本區塊的這個實體的任何的多個離開邊線，但是應該反而直接進到在這個軌跡的這個基本區塊實體的末端的該（刪除的）分支或跳躍的再收斂點。

若有來自一基本區塊實體的單一離開邊線，則跟隨邊線，無論是否有分支或跳躍。

若在選擇介於多數個離開邊線之間的此軌跡中的基本區塊實體的末端，具有一分支或間接跳躍，則遍歷（traversal）跟隨這多個離開邊線。

在一些實施例中，當在一軌跡中的遍歷遇到對這個軌跡含有一個或多個指令的一基本區塊實體，則將會有碼。可能使用已經存在池中的碼或是可能將新碼加入池。在任一事件中，待使用的碼被放在一特定位址。然後在這個路徑上最近產生的碼被固定成進到這個位址。可能有這個碼能夠連續放置在這個路徑上最近的在前碼之後的可能性。接著，不需要到此處，除非，最近的在前碼可能已成爲分支或跳躍。然後它的目標 IP 需要被安排在進到正確地方。這個路徑上最近的在前碼可能不是一分支或跳躍。在這個情況中，需要插入一無條件至正確目標的跳躍。

大部分基本區塊實體在軌跡中是典型地不可到達的。

產生的碼不需要具有（且應不具有）中間形式的大量無目的產生的靜態複本。產生的碼僅必須具有每一可到達的路徑的指令的正確序列。

當遍歷該中間形式中的一邊線時，可能自一個靜態複本進到另一個。在產生的碼中並不區別靜態複本。一般概念是盡可能方便地僅取得正確指令序列，例如，若已經擁有帶有正確指令序列碼，則關閉返回對於正確原始 IP 已經產生的碼的迴圈。另一例子是進到對於一不同靜態複本產生但具有正確指令序列的碼。

當進到已經在那的碼時，則發生問題。可能暫時存在正確的指令序列，但是接著完全不匹配。該碼在二種不同情況可能進到相同原始 IP，但是來自相同原始 IP 要求的碼序列於這二種情況是不同的。

線性靜態複製

在一些實施例中，線性靜態複製創造該碼的「複本」以避免控制流實際再加入於非迴圈分支的一般性再收斂點，直至下個返回邊線。這基本上直到包含迴圈的下一重複、或是該包含迴圈的退出。有助於成爲造成許多碼「複本」的一分支樹。

在大部分（但非全部）的例子中，在分支的一般性再收斂點之後已經保持分離的碼並不變得不同，除了對於不同軌跡的指令的不同子集（一期望差）。在碼產生中，這能夠一起放回（對於不同軌跡個別就不同指令子集），因

爲在一般性再收斂點，以及從那之後（永遠地）指令序列是相同的。複本已經消失。若並非全部的碼的可能的許多複本相同，非常可能陷入僅有少許不同可能性，故許多靜態複本實際上在產生的碼中僅造成少許靜態複本。

即使複本，對於一分支 B，全部離開並且產生的碼完整再收斂至精確如原始碼（除了對於不同軌跡的指令子集之外），則沒有來自這個靜態複製的益處不是真的。這個碼是用於傳送相依性的導管。若非分離的，則引起限制並行的錯誤相依性。必須分離它。除此之外，在 B 的一般性再收斂點碼的複本有時（雖然非經常）因軌跡分離而結果爲不同。

集體迴圈展開

在一些實施例中，集體迴圈展開創造對於巢套迴圈的碼的許多「複本」。舉例而言，若有 4 層次的巢套迴圈以及僅 2 路展開，則最內部迴圈主體有 16 複本。這些 16 複本極爲不可能全部結果爲不同。恰恰相反。一迴圈的展開具有遠少於 50%機會提供任何有用助益。對於一流大部分展開，及時常所有的展開是非生產性的。非生產性展開，大部分的展開，通常導致所有複本（對於迴圈）結果爲相同（除了用於不同軌跡的不同指令子集）。因此，大部分且時常是所有的展開於碼產生而被再次一起放回。但有時，少許複本爲不同且有益於並行（處理）。

若來自展開的迴圈主體的二個複本相同，則在碼產生

中，對於這迴圈的返回邊線會進到相同地方，因爲要求的跟隨指令序列永遠相同。這個迴圈的展開複本已經消失。若這是一內部迴圈，這在藉由外部迴圈創造的許多複本中以相同方式發生。

若一外部迴圈具有生產性展開，內部迴圈在外部迴圈的多個複本中並非不同是合理可能的，即使該外部迴圈的複本中有差異。迴圈自然地傾向於形成吊床。內部迴圈非常可能會成爲一子常式。將會僅有它的一個複本（除了對於不同軌跡的不同指令子集）。將會自一外部迴圈的殘留的多個複本呼叫。

線內程序

在一些實施例中，程序已被「線內」化，產生它們的「複本」。這是遞迴的，所以僅需少量呼叫層級以及少量呼叫行，就能夠有大量的「複本」。另一方面，程序爲DTSE子常式之一理想候選。可能地，程序的許多「複本」，在大部分共同情況中，它們結果全都是相同的（除了針對不同軌跡的不同指令子集）。或者，它們結果可能是少許實際上不同的版本（除了針對不同軌跡的不同指令子集）。因此，程序成爲一個或僅少數的DTSE子常式（除了針對不同軌跡的不同指令子集）。

程序，若它們不爲「線內」，可能造成錯誤相依性。因此，即使程序變成重組爲僅一個DTSE子常式（每一軌跡），仍期望對於相依性分析爲完整地「複製」。除此之

外，程序的「複本」有時，雖然非經常，因軌跡分離而結果爲不同。

複製儲存

極相同指令能夠最終出現在將要冗餘地執行的多個軌跡中。這因爲這個指令未被自多軌跡中刪除而發生。由於這能夠伴隨任何指令發生，其可爲出現在將要冗餘地執行的多個軌跡中的儲存。

在一些實施例中，DTSE 軟體標示相同儲存在多個軌跡中爲冗餘地的情況。該儲存可能取得一特有前綴或是可能在前加上一複製儲存標誌指令。在一些實施例中，一複製儲存標誌指令將會是於 DTSE 暫存器的儲存。該複製儲存標誌，無論採何種形式，必須指出其它哪些軌跡將要冗餘地執行這個相同儲存。

調準標誌

在一些實施例中，若 DTSE 硬體偵測在相同調準時距中來自超過一個的軌跡至相同位元組的儲存，將會宣告一違反並且引起到達最近的全域提交狀態的一狀態回復以及一流退出。當然，標示複製儲存除外。DTSE 硬體將會匹配冗餘地執行標示複製儲存並且將被作爲一單一儲存而提交。

時距標誌是調準時距分隔符。標示複製儲存是調準時距分隔符。調準標誌是調整時距分隔符。

在一些實施例中，調準標誌是一特有指令。它是於一DTSE 暫存器的一儲存並且指出其它哪些軌跡具有相同調準標誌。

若在多軌跡中有於相同位元組的儲存，硬體能夠適當地依程式序列放置這些儲存，規定衝突儲存在不同調準時距中。

DTSE 硬體得知來自相同軌跡的記憶體位址的程式序列。只要它們在不同調準時距中，硬體得知不同軌跡的記憶體位址的程式序列。在一些實施例中，若硬體發現一載入需要來自並非在相同軌跡中執行的儲存的資料的可能性，則將宣告一違反並且引起到達最近的全域提交狀態的一狀態回復以及一流退出。

在一些實施例中，DTSE 軟體將會在已經被看到命中相同位元組的多重軌跡中發生的儲存之間放置一些形式的調準標誌。DTSE 軟體將會放置調準標誌以使任何看到命中與儲存爲相同位址的任何載入將會適當地排序於硬體。

狀態儲存及回復

在一些實施例中，在各時距標誌設立有一全域提交點。時距標誌，其本身，發送一識別符至硬體。在一些實施例中，DTSE 軟體建立一表格。若必須回復狀態至最近的全域提交點，軟體將會自硬體取得該識別符並且在表格中查詢這個全域提交點。DTSE 軟體將會在表格中放置這個全域提交點的原始碼 IP 並與未頻繁改變且位在能夠在碼

預備時間得知的這個碼位置（例如其中該碼在運行的環）的其它狀態一起。其它資訊可能在可能也許已經自最近的全域提交點改變暫存器。很可能有一指標在此於軟體碼以回復狀態，由於這個碼可能對不同全域提交點而客製化。

在一些實施例中，加入碼至各時距標誌以儲存凡是需要儲存的資料以必要時能夠回復狀態。這很可能包括至少一些暫存器值。

在一些實施例中，可能客製化至全域提交點的碼被加入於回復狀態。對於該碼的指標被留跡（paced）於表格中。

全域提交點會相對頻繁地遭遇，但狀態回復則極少見。其優點在於：其以當必須執行實際狀態回復時之甚至大量增加的工作之成本，而最小化在全域提交點的工作。

從而，對於相依性分析與軌跡分離的一些實施例，碼全部散開至許多「複本」。在可執行碼產生時，大部分再次放回一起。

邏輯處理器管理

DTSE 可能實施於具有多同步多重緒硬體緒的一組核心，例如，每核心雙同步多重緒硬體緒。DTSE 系統能夠創造更多邏輯處理器以使各核心似乎具有，例如，四個邏輯處理器而非只有二個。加之，DTSE 系統能夠有效對於實施邏輯處理器管理核心資源。最終，若 DTSE 已分解一些碼串流成爲多重緒，這些緒能夠運行在邏輯處理器。

爲了在具有，例如，雙同步多重緒硬體緒的一核心上實行，例如，四個邏輯處理器，在一些實施例中，DTSE系統將對，例如，無法在核心硬體擁有狀態的二個邏輯處理器保持處理器狀態。DTSE 系統將不時切換各同步多重緒硬體緒的狀態。

DTSE 將對各軟體緒產生碼。DTSE 可能已經完成緒分解以自一單一原始碼串流創造出數個緒，或者 DTSE 可能自一單一原始碼串流僅創造一單一緒，依逐一準則。碼對於一單一原始碼串流以相同方式被產生，依任一方式。在軌跡分離時，碼可能分離成超過一個的緒，或是軌跡分離可能僅將所有碼放入相同單一軌跡。

在產生可執行碼之前，能夠靠該碼完成額外工作，包括指令的附加，以實行邏輯處理器管理。

在一些實施例中，DTSE 硬體將提供至少一個一致定址的儲存區位，但其（事實上）將對於執行存取的各同步多重緒硬體緒存取不同儲存器。在一實施例中，這是一處理器一般暫存器，諸如 RAX。這是於任何核心中而由運行在任何同步多重緒硬體緒所存取而作爲「RAX」而非儲存位置，因此該資料對於執行對「RAX」存取的每一同步多重緒硬體緒係爲不同。在一些實施例中，使用處理器一般暫存器運行程式碼，故 DTSE 需要一些 DTSE 硬體將提供的其它的同步多重緒硬體緒特有儲存器。這可能是，例如，在 DTSE 邏輯模組中每同步多重緒硬體緒一個或少數暫存器。

特別地，在一些實施例中，一同步多重緒硬體緒特有儲存器暫存器（ME）將含有對於現正運行在這個同步多重緒硬體緒的邏輯處理器的一指標於狀態儲存表格。該表格在這個區位將必然包含其它資訊，諸如一於下個要運行的邏輯處理器的儲存區域的指標以及一於先前運行在這個同步多重緒硬體緒儲存表格的邏輯處理器的指標。

對於所有緒，DTSE 對所有原始碼串流所產生的所有碼在相同位址空間中。因此對於任何原始碼串流的任何產生的碼，能夠跳躍至對於任何原始碼串流的任何產生的碼。DTSE 特有資料也是在相同位址空間中。該程式資料空間（一般而言）對各原始碼串流是在不同位址空間中。

有效緒切換

在一些實施例中，DTSE 將在對其產生碼的各緒中插入 HT 切換入口點與退出點。從而，這樣的入口點的使用已在硬體段落討論了。

HT 切換入口點

在一些實施例中，在 HT 切換入口點的碼將自 ME 讀取，到它擁有的儲存表格的一指標並且接著該指標到下個邏輯處理器儲存表格。從這個表格，能夠取得下個 HT 切換入口點的 IP 以進到接著要進行的入口點。碼能夠使用一特有指令，其將會將這個位址推到分支預測器的返回預測堆疊之上。選擇性地，可能在這個位址以及也許在額外

位址發佈一預取（prefetch）。這都是對於將在此現有 HT 切換入口點之後完成的下個 HT 切換的設定。返回預測器需要立刻準備，如此將會正確的預測下個 HT 切換。若在下個 HT 切換之後可能有 I 快取失誤，應該在這個點發佈預讀，以使得在下個 HT 緒切換的 I 快取中 I 串流。該碼將接著在這個點自它擁有的儲存表格讀取它的需求狀態，並且在這個 HT 切換入口點之後重返執行該碼。當這需要時，能夠包括載入 CR3、EPT、及片段暫存器。有助於使得邏輯處理器分享相同同步多重緒硬體緒具有相同位址空間，例如，因爲它們全運行來自相同處理的緒，故並非必要在 HT 切換時重載這些暫存器，縱使沒有必要。

HT 切換退出點

在一些實施例中，在 HT 切換退出點的碼將自 ME 讀取到它擁有的儲存表格的一指標。將儲存對於重返它擁有的儲存表格所必須的狀態。接著將會自它擁有的儲存表格讀取到下個要運行的邏輯處理器的儲存表格的一指標並且將其寫入至 ME。讀取要進到的下個 HT 退出點的 IP，並且將這推上堆疊。進行一返回指令以執行到該必須的 HT 切換入口點的一完整預測跳躍。

應注意，在 HT 切換退出點的碼係控制有當再次取得一同步多重緒硬體緒之 IP，並運行之。在它擁有的儲存表格的 IP 中能夠放入任何想要的東西。

有效不可預料的間接分支

在一些實施例中，一不可預料的間接分支可藉由DTSE改變間接分支而僅計算該分支目標而有效完成。它緊接著一 HT 切換退出點，但所計算之該分支目標被儲存於儲存表格。

當切換回這個緒中，將自然地進到該間接分支的正確目標。此對於任一間接分支或 HT 切換，能夠在無分支失誤預測以及沒有 I 快取失誤而完成。

切換資源於邏輯處理器

在一些實施例中，直至分支報告，有一特有指令或前綴（prefix），稱爲停止提取（Stop Fetch）。這個指令能夠緊接地插入在分支或間接跳躍之前。

當解碼到直至分支報告的停止提取時，對此 I 串流的指令提取將爲停止且將不會對這 I 串流的下個緊接的指令的指令解碼，假設其它同步多重緒硬體緒仍在進行正在進行。若其它同步多重緒硬體緒不在進行，則忽略這個指令。接著的指令應該是一分支或間接跳躍。對它上標籤。在分支與跳躍在被正確地預測或失誤預測執行時將爲報告。當標籤的分支爲報告時，對於這 I 串流的指令提取以及解碼被恢復。當在這個同步多重緒硬體緒中的任何分支報告一失誤預測時，指令提取及解碼被恢復。

在一些實施例中，直至載入報告之前，有一特有指令或前綴（prefix），稱爲停止提取（stop fetch）。這個指

令有時能夠緊接地插入在載入之後。它具有一運算元，將會被執行而爲該載入的結果。停止提取爲實際執行直到載入報告指令。其在執行時將報告而不會被取消。直至載入報告指令的停止提取有二種形式，有條件的與無條件的。

當被解碼時，直至載入報告指令的無條件停止提取將會停止指令提取以及解碼。只有在其它同步多重緒硬體緒正在進行時，當被解碼時，直至載入報告指令的條件性停止提取將會停止在這 I 串流的指令提取以及解碼。當該指令報告未取消執行時，二種形式的指令恢復這 I 串流的指令提取及解碼，並且對於這 I 串流沒有未解決之資料快取失誤。

碼分析

快閃概析將指出各個別分支或跳躍執行實體，若這個執行實體被失誤預測或正確地預測。將會指出取得 I 快取失誤、第二階快取失誤、以及於 DRAM 的失誤的指令執行實體。若執行實體取得一資料快取失誤、第二階快取失誤、或是於 DRAM 的失誤時，將會對各載入指出這個執行實體。

DTSE 軟體所作的所有形式的靜態複製也可能使用於邏輯處理器管理。在一些實施例中，載入、分支、及間接跳躍的所有靜態實體取得失誤號碼。指令的靜態實體在這些實施例中取得提取快取失誤號碼。

（由原始 IP（所指到的））相同指令的不同靜態實體

非常頻繁地具有非常不同失誤行爲，因此一般最好使用指令的靜態實體。一指令的實體愈多，對於各實體的失誤率數目之較佳機會將會高或低。中間失誤率數目較難以處理。

儘管盡了一切努力且雖然對比於僅使用 IP 有許多改善，可能將仍將有許多指令實體爲中間程度失誤次數（number）。群組化在一些實施例中是控制中間程度失誤次數的一種方式。各具有一中間程度失誤預測率的分支的一小樹可提供在經過該樹的執行路徑上的某處之一些失誤預測的大可能性。相似地，各帶有一中間程度快取失誤率的數個載入的一連續字串能夠在至少一個載入提供失誤的大可能性。

迴圈展開是一群組化機制。在該迴圈之一重複中的一個別載入可能具有一中間程度快取失誤率。若將該載入的一些數量執行而反覆一些數量的迴圈重複視爲一群組，則在至少一個此些重複中會提供一高可能性的快取失誤。一重複內的多個載入自然地與群組化多重重複群組在一起。

在一些實施例中，DTSE 軟體創造群組而使各群組具有一相對高可能性的某種失誤。群組有時能夠壓縮。特別是對分支樹。分支樹中的較晚分支能夠藉由將從前習慣爲在分支之前而現在爲在分支之後的靜態複製指令前移。此將該分支更緊密地與該樹一起包裝。

若群組僅很可能取得一分支失誤預測，通常不值得 HT 切換。在一些實施例中，在路徑上之剛好最後群組分

支之前的群組之外的路徑上插入直至分支報告的停止提取。執行路徑上的群組中的分支將被解碼並且然後只要其它同步多重緒硬體緒正在進行，則其會停止解碼。這提供核心資料給其它同步多重緒硬體緒。若群組中沒有失誤預測，當執行路徑上最後的群組分支爲報告時，將再次開始提取與解碼。否則，一經分支爲報告初一失誤預測，提取就將會在改正的目標位址重啓。這並非相當理想，因爲分支可能並未依序報告。

然而，HT 切換爲使用於具有高可能性失誤預測的間接分支，如所述般。

相似地，若群組僅很可能取得資料快取失誤，通常最好不進行 HT 切換。可能的話，群組中的載入在一些實施例中將被移動以使全部的載入是在任何載入的第一消費者之前。在一些實施例中是使直至載入報告指令的條件性停止提取爲相依於群組中的最後載入並且放置在該載入之後但在任何消費者之前。

直至載入報告指令的無條件停止提取可被使用，若資料快取資料快取失誤幾乎必然，但其僅爲一資料快取失誤。

群組中的頻繁載入通常不被放在任何消費者之前。舉例而言，若該群組爲迴圈的展開重複，則其不起作用。在這情況中，最好是使得群組足夠大而使至少一個或是最好數個資料快取失誤幾乎無可避免。若群組爲迴圈的展開重複，則通常能夠達成。在一些實施例中，產生一組預讀以

涵蓋該群組中的載入。首先放置預取，然後 HT 切換，並且再來是該碼。

一個帶有高可能性的第二階快取失誤的一群組，D 串流或 I 串流使其正當（justifies）且 HT 切換。首先放置預取，然後 HT 切換，並且再來是該碼。

平均約 30%機會的失誤於 DRAM 能夠使其正當一 HT 切換。在這些例子中，在一些實施例中，首先完成預取，然後 HT 切換。若能夠涵蓋數個失誤，最好仍是群組化更多以取得較高且較好的失誤可能性。

在一些實施例中，在其它同步多重緒硬體緒的工作爲「正在涵蓋（covering）」，同時發生 HT 切換。其目的在於始終使一個同步多重緒硬體緒進行即時（real）的工作。

若一個同步多重緒硬體緒在其它停止提取時進行即時工作，則在工作中的同步多重緒硬體緒中隨時有風險的問題。故通常不會長時間僅依靠單一工作中的同步多重緒硬體緒。另外，典型地不希望長時間的停止提取。若要爲長，則在一些實施例中是作 HT 切換，使得工作中的同步多重緒硬體緒由另一個爲備份，用於在遭遇阻礙時。

範例運算系統及處理器

圖 12 爲繪示依據本發明之實施例的一核心之範例非序架構的一方塊圖。然而上述指令也可能在有序架構中實行。圖 12 中，箭號代表二個以上單元之間的連接，箭號

方向指出這些單元之間的資料流方向。這個架構的組成可用於處理之前所詳述的指令，包括提取、解碼、及執行這些指令。

圖 12 包括一前端單元 1205 連接到一執行引擎單元 1210 及一記憶單元 1215；執行引擎單元 1210 進一步連接到記憶單元 1215。

前端單元 1205 包括一 1 階（L1）分支預測單元 1220 連接到一 2 階（L2）分支預測單元 1222。這些單元允許核心提取及執行指令而不需等待分支被解析。L1 與 L2 號的預測單元 1220 及 1222 連接於一 L1 指令快取單元 1224。L1 指令快取單元 1224 藉由執行引擎單元 1210 保持指令或有可能被執行的一個或多個緒。

L1 指令快取單元 1224 連接於一指令轉譯旁看緩衝器（ITLB）1226。ITLB 1226 連接於一將位元組串流劃分爲離散指令的指令提取及預解碼單元 1228。

指令提取及預解碼單元 1228 連接於一指令佇列單元 1230 以儲存這些指令。一解碼單元 1232 對包括上述指令的佇列指令予以解碼。在一些實施例中，解碼單元 1232 包含一複雜解碼器單元 1234 以及三個簡單解碼器單元 1236、1238、及 1240。簡單解碼器能夠處理大部分（若非全部）x86 指令，其經解碼成爲單一 uop（微運算操作）。複雜解碼器能夠解碼映射至多個微運算操作的指令。解碼單元 1232 也可能包括一微代碼 ROM 單元 1242。

L1 指令快取單元 1224 進一步連接於記憶單元 1215

中的一 L2 快取單元 1248。指令 TLB 單元 1226 進一步連接於記憶單元 1215 中的一第二階 TLB 單元 1246。解碼單元 1232、微代碼 ROM 單元 1242、及一迴圈串流偵測器（LSD）單元 1244 各連接於執行引擎單元 1210 中的一更名/配置單元 1256。LSD 單元 1244 偵測何時一軟體迴圈被執行，偵測停止預測分支（以及潛在非正確的預測該迴圈的最後的分支），以及自它脫離的串流指令。在一些實施例中，LSD 1244 貯藏微操作指令。

執行引擎單元 1210 包括連接於一引退單元 1274 與一統一排程單元 1258 的更名/配置單元 1256。更名/配置單元 1256 決定資源需求優先於任何暫存器更名並且指派可用的資源用於執行。這單元也更名邏輯暫存器至實體暫存器檔案的實體暫存器。

引退單元 1274 進一步連接於執行單元 1260 並且包括一重定序緩衝單元 1278。這個單元使指令在完成之後引退。

統一排程單元 1258 進一步連接於一與執行單元 1260 連接的實體暫存檔案單元 1276。這個排程器在運行於處理器上的不同緒之間共享。

實體暫存器檔案單元 1276 包含一 MSR 單元 1277A、一浮點暫存器單元 1277B、及一整數暫存器單元 1277C，並且可包括未顯示的額外暫存器檔案（例，混疊於 MMX 包整之整數平面暫存器檔案 550 上的純量浮點堆疊暫存器檔案 545）。

執行單元 1260 包括三個混合純量及 SIMD 執行單元 1262、1264、及 1272；一載入單元 1266；一儲存位址單元 1268；一儲存資料單元 1270。載入單元 1266、儲存位址單元 1268、及儲存資料單元 1270 執行載入/儲存及記憶操作並且各進一步連接於記憶單元 1215 中的一資料 TLB 單元 1252。

記憶單元 1215 包括連接於資料 TLB 單元 1252 的第二階 TLB 單元 1246。資料 TLB 單元 1252 連接於一 L1 資料快取單元 1254。L1 資料快取單元 1254 進一步連皆於一 L2 快取單元 1248。在一些實施例中，L2 快取單元 1248 進一步連接於在記憶單元 1215 內部及/或外部的一 L3 以上的快取單元 1250。

接下來是適合執行詳述於此的指令的範例系統。在用於膝上型電腦、桌上型電腦、手持式 PC、個人數位助理、工程工作站、伺服器、網路裝置、網路集線器、交換器、嵌入式處理器、數位訊號處理器（DSP）、繪圖裝置、視訊遊戲裝置、機上盒、微控制器、行動電話、攜帶式媒體播放器、手持式裝置、及其它各式電子裝置的技術中所知的其它系統設計及配置也是適合的。通常，能併入此處所揭示之處理器及/或其他執行邏輯之種類繁多的系統或電子裝置係普遍適合的。

參閱圖 13，所示爲根據本發明的一個實施例的一系統 1300 的方塊圖。系統 1300 可包括一個或多個處理元件 1310、1315，其連接於繪圖記憶體控制器中樞（GMCH）

1320。額外的處理元件 1315 的選擇性類別在圖 13 中以虛線表示。

各處理元件可爲單核心或是可，改以，包括多核心。處理單元可，選擇性地，包括除了處理核心之外的其它晶粒上元件，諸如整合記憶控制器及/或整合 I/O 控制邏輯。同樣地，就至少一個實施例而言，處理元件的核心可能爲多重緒以致每核心可能包括多於一個硬體緒上下文。

圖 13 繪示 GMCH 1320 可能連接於可能爲，例如，動態隨機存取記憶體（DRAM）的記憶體 1340。DRAM 可能，就至少一個實施例而言，與非揮發性快取相關連。

GMCH 1320 可爲晶片組、或是晶片組的一部分。GMCH 1320 可與處理器 1310、1315 溝通以及控制處理器 1310、1315 與記憶體 1340 之間的互動。GMCH 1320 也可運作如同處理器 1310、1315 與系統 1300 的其它元件之間的加速匯流排介面。就至少一個實施例而言，GMCH 1320 通過一多落點匯流排而與處理器 1310、1315 溝通，諸如一前端匯流排（FSB）1395。

更者，GMCH 1320 連接於一顯示器 1345（諸如一平板顯示器）。GMCH 1320 可包括一整合繪圖加速器。GMCH 1320 進一步連接於一輸入/輸出（I/O）控制器中樞（ICH）1350，其可用於將各種周邊裝置連接至系統 1300。在圖 13 的實施例中所例示的是一外部繪圖裝置 1360，其可爲與另一周邊裝置 1370 一起連接於 ICH 1350 的一離散繪圖裝置。

選擇性地，在系統 1300 中也可出現附加的或是不同的處理元件。舉例而言，附加處理元件 1315 可包括與處理器 1310 相同的附加處理器，異質或非對稱於處理器 1310 的附加處理器，加速器（諸如，例，繪圖加速器或數位訊號處理（DSP）單元），場式可編程閘極陣列，或任何其他處理元件。在實體資源 1310、1315 之間，就包括架構、微架構、熱、能耗特徵等的優點的測度的值譜方面，能夠有各種差異。這些差異可有效彰顯它們做爲在處理元件 1310、1315 之中的非對稱性及異質性。就至少一個實施例而言，各種處理元件 1310、1315 可存在相同晶粒包整程式中。

參閱圖 14，所示爲根據本發明之一實施例的一第二系統 1400 的方塊圖。如圖 14 所示，多處理器系統 1400 爲點對點互連系統，並包括一第一處理元件 1470 與一第二處理元件 1480 通過點對點介面 1450 而連接。如圖 14 所示，各處理元件 1470 及 1480 可爲多核心處理器，包括第一與第二處理器核心（即，處理器核心 1474a 與處理器核心 1474b 以及處理器核心 1484a 及 1484b）。

可選擇地，處理元件 1470、1480 之一個或多個可爲有別於處理器的一元件，諸如一加速器或一場式可編程閘極陣列。

儘管只有顯示二個處理元件 1470、1480，當知本發明並不作如此限制。在其它實施例中，在一給定的處理器中可出現一個或多個附加的處理元件。

第一處理元件 1470 可進一步包括一記憶體控制器中樞（MCH）1472 以及點對點（P-P）介面 1476 及 1478。相似地，第二處理元件 1480 可包括一 MCH 1482 以及 P-P 介面 1486 及 1488。處理器 1470、1480 可使用 PtP 介面電路 1478、1488 而通過點對點（PtP）介面 1450 交換資料。如圖 14 所示，MCH 1472 及 1482 使處理器連接個別的記憶體，名爲記憶體 1442 及記憶體 1444，其可爲局部附接於個別處理器的部分主記憶體。

處理器 1470、1480 各自可使用點對點介面電路 1476、1494、1486、1498 而經由個別的 PtP 介面 1452、1454 與晶片組 1490 交換資料。晶片組 1490 也可經由一高性能繪圖介面 1439 而與一高性能繪圖電路 1438 交換資料。本發明的實施例可設在任何具有任何數量的處理核心的處理器內，或是圖 14 的各 PtP 匯流排代理內。在一個實施例中，任何處理器核心可包括，或是關聯於，一局部快取記憶體（圖未示）。更者，任一處理器可包括一共享快取（圖未示）在兩處理器的外部，該共享快取卻經由 p2p 互連而與該處理器連接，以至於若處理器屬於一低功率模式，可將處理器任一或二者的局部快取資料儲存在該共享快取中。

第一處理元件 1470 及第二處理元件 1480 可分別經由 P-P 互連 1476、1486 及 1484 而連接於一晶片組 1490。如圖 14 所示，晶片組 1490 包括 P-P 介面 1494 及 1498。更者，晶片組 1490 包括一介面 1492 以將晶片組 1490 與一

高性能繪圖引擎 1448 連接。在一個實施例中，匯流排 1449 可用於將繪圖引擎 1448 連接晶片組 1490。或者，一點對點互連 1449 可連接這些構件。

接著，晶片組 1490 可經由一介面 1496 而連接於一第一匯流排 1416。在一個實施例中，第一匯流排 1416 可爲一周邊構件互連（PCI）匯流排，或是諸如 PCI Express 匯流排或是另一第三代 I/O 互連匯流排的匯流排，然而本發明的範圍並不侷限於此。

如圖 14 所示，各式 I/O 裝置 1414 可與將第一匯流排 1416 連接於第二匯流排 1420 的一匯流排橋接器 1418 一起連接於第一匯流排 1416。在一個實施例中，第二匯流排 1420 可爲一低針腳數（LPC）匯流排。各式裝置可連接於包括，例如，一鍵盤/滑鼠 1422、通訊裝置 1426、及諸如磁碟機或是可包括碼 1430（在一個實施例中）的其他大量儲存裝置的資料儲存單元 1428。再者，一音頻 I/O 1424 可連接於第二匯流排 1420。注意，其它架構也是可能的。例如，除了圖 14 的點對點架構之外，一系統可實施一多落點匯流排或是其它如此架構。

現在參閱圖 15，所示爲根據本發明之一實施例的一第三系統 1500 的方塊圖。圖 14 與 15 中的相似元件帶有相似的元件符號，並且圖 14 一些確鑿的部分在圖 15 省略，以免使其它部分在圖 15 變得不顯著。

圖 15 繪示處理元件 1470、1480 可分別包括整合記憶體及 I/O 控制邏輯（「CL」）1472 及 1482。就至少一個

實施例而言，CL 1472、1482 可包括記憶體控制中樞邏輯（MCH），例如上述般與圖 13 及 14 連接。此外。CL 1472、1482 也可包括 I/O 控制邏輯。圖 15 繪示不僅是記憶體 1442、1444 連接 CL 1472、1482，並且 I/O 裝置 1514 也連接於控制邏輯 1472、1482。舊有 I/O 裝置 1515 連接於晶片組 1490。

於此所揭示的這些機制的實施例可實現於硬體、軟體、韌體、或是這類實現手段的結合中。本發明的實施例可實現爲電腦程式或是爲包括至少一個處理器、一資料儲存系統（包括揮發性及非揮發性記憶體及/或儲存元件）、至少一個輸入裝置、及至少一個輸出裝置的可編程系統上執行的程式碼。

程式碼，例如圖 14 中所繪示的碼 1430，可應用於輸入資料以執行於此所述功能並且產生輸出資訊。輸出資訊可應用於一個以上的輸出裝置，在已知方式中。就本申請案的意圖來看，處理系統包括任何具有一處理器的系統，諸如，例如；一數位訊號處理器（DSP）、一微控制器、一特殊應用積體電路（ASIC）、或一微處理器。

程式碼可實現在一高階程序或物件指向程式語言以與處理系統溝通。若有需要，程式碼也可實現在組合或機器語言。事實上，於此所述機制並不將範圍限制於任何特定的程式語言。在任何情況中，該語言可爲一編譯或直譯語言。

至少一個實施例的一個或多個態樣可藉由儲存在機器

可讀媒體上的代表性資料（其表示在處理器內的各式邏輯）而實現，其在機器讀取時會使機器建立邏輯以執行於此所述技術。被稱爲「IP 核心」的這類代表物可儲存在一有形機器可讀媒體以及供應至各種消費者或是製造設施以載入實際做成邏輯或處理器的建造機器中。

這類機器可讀儲存媒體可包括，但不限制於，非暫態有形排列的粒子，其藉機器或裝置所製造或形成，包括儲存媒體，諸如硬碟、任何其他類型的磁碟（包括軟碟、光碟、光碟唯獨記憶體（CD-ROM）、可覆寫光碟（CD-RW）、及磁光碟）、半導體裝置（諸如唯讀記憶體（ROM）、諸如動態隨機存取記憶體（DRAM）、靜態隨機存取記憶體（SRAM）、可擦寫程式化唯讀記憶體（EPROM）、快閃記憶體、電子可擦寫程式化唯讀記憶體（EEPROM）、磁或光卡之類的隨機存取記憶體（RAM）、或是適合儲存電子指令的任何其他類型的媒體）。

因此，本發明之實施例也包括含有指令或含有諸如HDL 之設計資料的非暫態有形機器可讀媒體，其界定於此所述的結構、電路、設備、處理器及/或系統特徵。這樣的實施例也可成爲程式產物。

於此所揭露的指令的某些作業可藉由硬體構件來執行並且可體現於機器可執行指令，而用於引起，或至少結果爲，與指令一同編程的電路或其他硬體構件執行作業。該電路可包括一般用途或特殊用途處理器、或邏輯電路，以僅命名少量例子。該作業也可選擇性地藉由硬體與軟體之

組合而執行。執行邏輯及/或一處理器可包括對一機器指令或源自機器指令的一個或多個控制訊號有所反應的特有或特殊電路或其它邏輯，以儲存一指令特定結果運算元。例如，於此揭露的指令的實施例可於圖 13、14 及 15 之一或多個系統中被執行，且指令的實施例可儲存於系統中待執行的程式碼中。

上述說明意圖闡明本發明的較佳實施例。由上述討論，也應是可輕易得知的，特別在這樣成長快速並且不易預見進一步進展的技術領域中，在該領域熟習此項技術者能夠對本發明在排列與細節上進行修改，而不違背在隨附專利範圍及其等效物之範圍內的本發明的原則。例如，方法的一個或多個作業可結合或進一步拆解。

替代實施例

雖然已經說明能夠自然執行於此所述的指令的實施例，本發明的替代實施例可透過運行在執行不同指令組的處理器的一仿真層而執行（例，執行加州森尼維耳市 MIPS 科技的 MIPS 指令組的處理器，執行加州森尼維耳市 ARM 國際科技的 ARM 指令組的處理器）。同時，雖然圖式中的流圖顯示本發明某些實施例所執行的作業的特定順序，當知這樣的順序是範例（例，替代實施例可依不同順序執行該作業、結合某些作業、使作業部分同時發生...等）。

在上面的說明中，爲了說明之故，依序列舉出許多特有細節以提供本發明之實施例的徹底理解。顯而易見的，

對該領域熟習此項技術者而言，可實行一個或多個其它實施例而無須這些特有細節。所述的特定實施例並非用以限制本發明而是在闡明本發明的實施例。本發明的範圍並非由上述所提出的特有例子而是僅由下列的專利範圍來決定。

【圖式簡單說明】

本發明以例示方式繪示而不受限於隨附圖式，相似元件以類似元件符號標示，其中：

圖 1 圖表繪示動態緒切換執行架構作業之一範例實施例。

圖 2 繪示依據一些實施例的 DTSE 作業之一範例方法。

圖 3 繪示一 DTSE 架構之一實施例。

圖 4 繪示依據一些實施例的用於包整程式的主要硬體區塊。

圖 5 繪示根據一實施例的時距執行。

圖 6 繪示 DTSE 硬體之一實施例的一更詳細圖式。

圖 7 繪示根據一些實施例的 XGC 之應用。

圖 8-11 繪示一些軟體作業之範例。

圖 12 爲繪示依據本發明之實施例的一核心之範例非序架構的一方塊圖。

圖 13 顯示根據本發明之一個實施例的一系統之方塊圖。

圖 14 顯示根據本發明之一實施例的一第二系統之方塊圖。

圖 15 顯示根據本發明之一實施例的一第三系統之方塊圖。

【主要元件符號說明】

301：核心

303：全域記憶體一致性

305：軟體用資訊擷取

307：全域引退

309：全域暫存器狀態

311：軟體

401：核心

403：違反偵測、基元性提交、熱 IP 偵測、及概析邏輯

405：中階快取

409：XMC Guard 快取

601：核心

602：序列緩衝器（OB）

603：推測載入資料快取（SLC）

605：載入儲存序列緩衝（LSOB）

607：推測儲存快取（SSC）

609：斷言

611：儲存正確性快取（SCC）

613：載入正確性快取（LCC）

615：寫入結合快取（WCC）

617：中階快取（MLC）

1205：前端單元

1210：執行引擎單元

1215：記憶單元

1220：L1 分支預測單元

1222：L2 分支預測單元

1224：L1 指令快取單元

1226：指令轉譯旁看緩衝器

1228：指令提取及預解碼單元

1230：指令佇列單元

1232：解碼單元

1234：複雜解碼器單元

1236：簡單解碼器單元

1238：簡單解碼器單元

1240：簡單解碼器單元

1242：微代碼 ROM 單元

1244：迴圈串流偵測器單元

1246：第二階 TLB 單元

1248：L2 快取單元

1252：資料 TLB 單元

1254：L1 資料快取單元

1256：更名/配置單元

1258：排程單元

1260：執行單元

1262：混合純量及 SIMD 執行單元

1264：混合純量及 SIMD 執行單元

1266：載入單元

1268：儲存位址單元

1270：儲存資料單元

1272：混合純量及 SIMD 執行單元

1274：引退單元

1276：實體暫存器檔案單元

1277A：MSR 單元

1277B：浮點暫存器單元

1277C：整數暫存器單元

1278：重定序緩衝單元

1300：系統

1310：處理元件

1315：處理元件

1320：繪圖記憶體控制器中樞（GMCH）

1340：記憶體

1345：顯示器

1350：輸入/輸出控制器中樞

1360：外部繪圖裝置

1370：周邊裝置

1395：前端匯流排

1400：系統

1414：I/O 裝置

1415：晶片組

1416：匯流排

1418：匯流排橋接器

1420：匯流排

1422：鍵盤/滑鼠

1424：音頻 I/O

1427：通訊裝置

1428：資料儲存單元

1430：碼

1432：記憶體

1434：記憶體

1439：高性能繪圖介面

1450：點對點介面

1452：PtP 介面

1454：PtP 介面

1470：處理器

1472：記憶體控制器中樞

1474a：處理器核心

1474b：處理器核心

1476：點對點介面

1478：點對點介面

1480：處理器

1482：記憶體控制器中樞

1484a：處理器核心

1484b：處理器核心

1486：點對點介面

1488：點對點介面

1490：晶片組

1492：介面

1494：點對點介面

1496：介面

1498：點對點介面

1500：系統

1514：I/O 裝置

1515：舊有 I/O 裝置

七、申請專利範圍：

1. 一種方法，包括：

在一第一處理器核心執行原始源碼；

將一第二處理器核心設置爲在一偵測階段，其中在該偵測階段中，該第二處理器核心係爲偵測一指示值而切換成與該第一處理器核心協作之執行模式；

在該第一處理器核心中概析該原始源碼；

在該第二處理器核心中自該原始源碼產生協作碼，以由該第一處理器核心與該第二處理器核心使用該概析來協作執行，其中該協作碼爲該原始源碼之緒版本，伴隨可能的入口點，一入口點爲對應於動態執行碼之一大部分的點；

偵測一指示值而在該第二處理器核心中切換成協作執行模式；

在該第一處理器核心與該第二處理器核心中執行該產生的協作碼。

2. 如申請專利範圍第 1 項的方法，更包括：當命中該指示值而切換時，提供該第一處理器核心進入一不同的執行模式。

3. 如申請專利範圍第 1 項的方法，其中概析該原始源碼包含收集針對一設定的指令量之有關載入、儲存及分支之資訊。

4. 如申請專利範圍第 1 項的方法，更包括：當該產生的協作碼成功完成時，在該第一與第二處理器核心中停

止該產生的協作碼的執行。

5. 如申請專利範圍第 1 項的方法，其中在該第一與第二處理器核心中執行該產生的協作碼包含：

分離執行兩個緒；

使用包整程式硬體以緩衝記憶體載入及儲存；

查對該緩衝的記憶體載入及儲存之可能的違反；以及

基元性提交一狀態以在維持記憶體排序的同時提供前移處理。

6. 如申請專利範圍第 5 項的方法，更包括：

當違反時，在該第一與第二處理器核心停止該產生的協作碼的執行，並退至最近的提交點。

7. 如申請專利範圍第 6 項的方法，更包括：

當違反時在該第一與第二處理器核心停止該產生的協作碼的執行時，在該第一處理器核心執行該原始源碼，並將該第二處理器核心設置回一偵測階段，其中在該偵測階段中，該第二處理器核心係爲偵測一指示值而切換成與該第一處理器核心不同且協作之執行模式。

8. 一種裝置，包含：

第一及第二處理核心，用以當偵測到原始碼中的熱碼入口點時執行協作碼，其中該協作碼爲該原始碼之緒版本，伴隨可能的入口點，其中該第一處理核心係爲概析該原始碼的碼之熱區域且該第二處理核心係爲偵測該原始碼之熱區域，其中碼之熱區域爲對應於該碼之動態執行之大部分的碼之一部分，且從該概析的碼產生該協作碼；及

硬體包整程式，用以緩衝由該第一及第二處理核心所執行之記憶體載入及儲存，查對該緩衝的記憶體載入及儲存之可能的違反，以及基元性提交一狀態以在維持記憶體排序的同時提供前移處理。

9. 如申請專利範圍第 8 項的裝置，更包括：中階快取，用以合併該協作碼的執行狀態。

10. 如申請專利範圍第 8 項的裝置，更包括：末階快取。

11. 如申請專利範圍第 8 項的裝置，其中該硬體包整程式用以當放棄時，廢棄該緩衝的記憶體載入及儲存。

12. 如申請專利範圍第 11 項的裝置，其中該放棄係發現於儲存/儲存違反時。

13. 如申請專利範圍第 11 項的裝置，其中該放棄係發現於載入/儲存違反時。

14. 如申請專利範圍第 11 項的裝置，其中當該放棄時，該第一處理核心退至最近的提交點。

15. 如申請專利範圍第 8 項的裝置，其中該第一處理核心用以執行該原始碼直到到達該入口點。

16. 如申請專利範圍第 8 項的裝置，其中在當命中指示值而切換，該硬體包整程式已概析該原始碼以進入不同的執行模式之後，提供該第一處理核心。

17. 如申請專利範圍第 8 項的裝置，其中該第一處理核心係藉由收集針對一設定的指令量之有關載入、儲存及分支之資訊而概析該原始碼。

18. 如申請專利範圍第 8 項的裝置，其中使用一硬體表偵測該碼之該熱區域，以偵測經常被存取的熱區域，且該熱碼入口點爲指令指標。

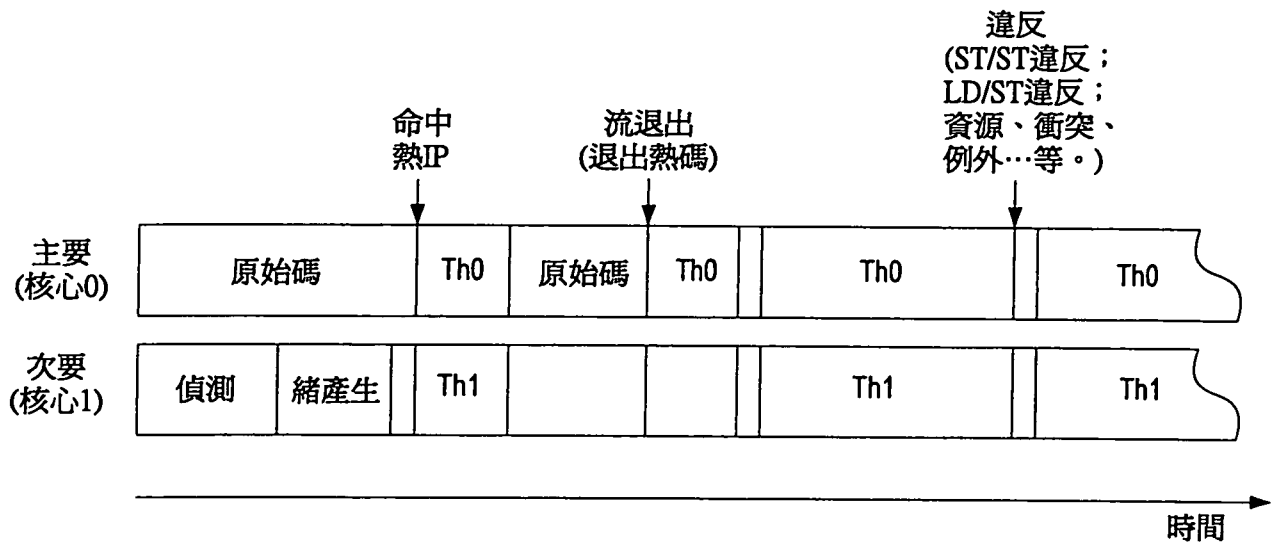
違反
(ST/ST違反；
LD/ST違反；
資源、衝突、
例外…等。)
命中
熱IP
流退出
(退出熱碼)
主要
(核心0)
原始碼
Th0
原始碼
Th0
Th0
Th0
次要
(核心1)
偵測
緒產生
Th1
Th1
Th1
時間

圖1

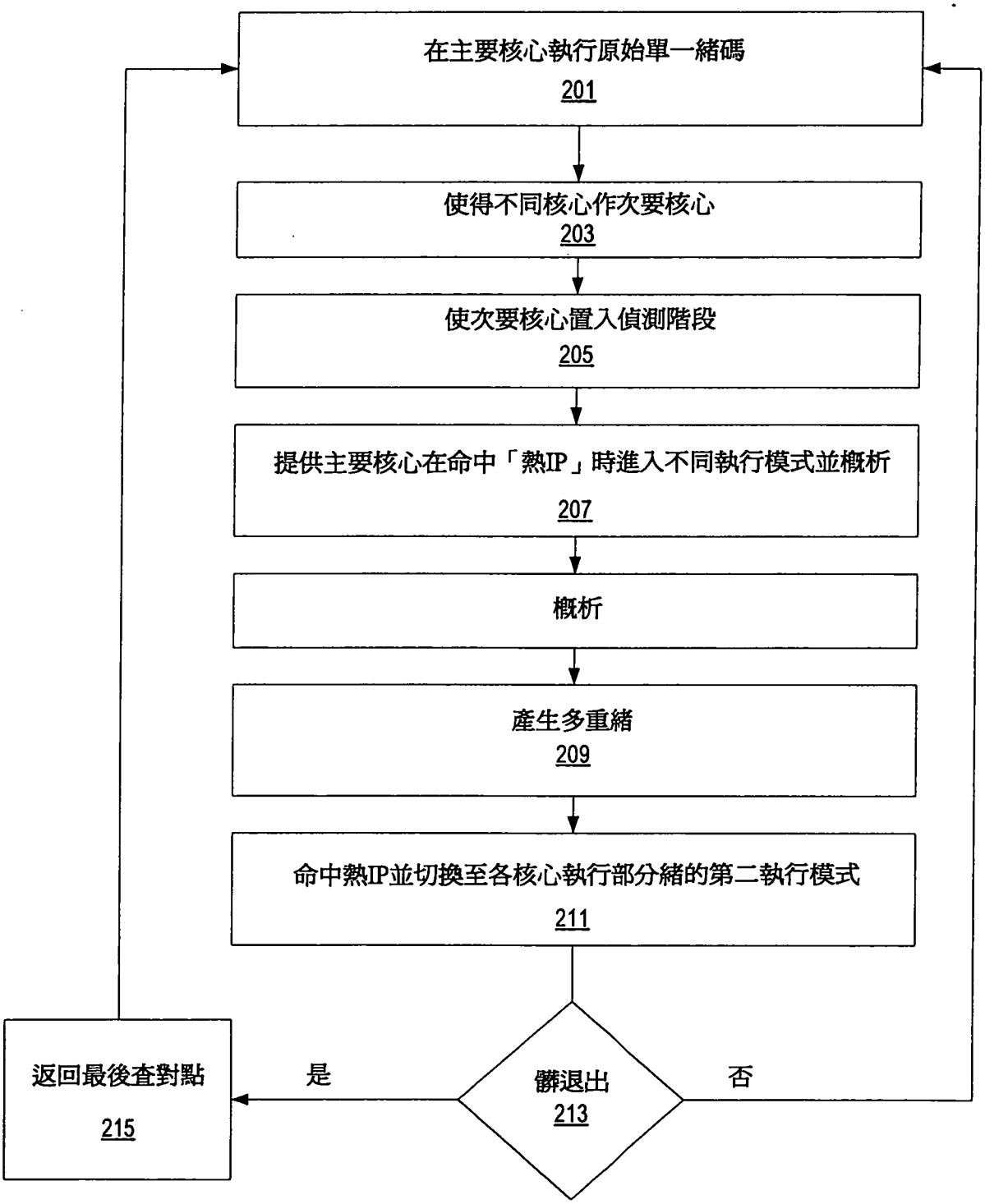
在主要核心執行原始單一緒碼
201
使得不同核心作次要核心
203
使次要核心置入偵測階段
205
提供主要核心在命中「熱IP」時進入不同執行模式並概析
207
概析
產生多重緒
209
命中熱IP並切換至各核心執行部分緒的第二執行模式
211
罷退出
213
是
否
返回最後查對點
215

圖2

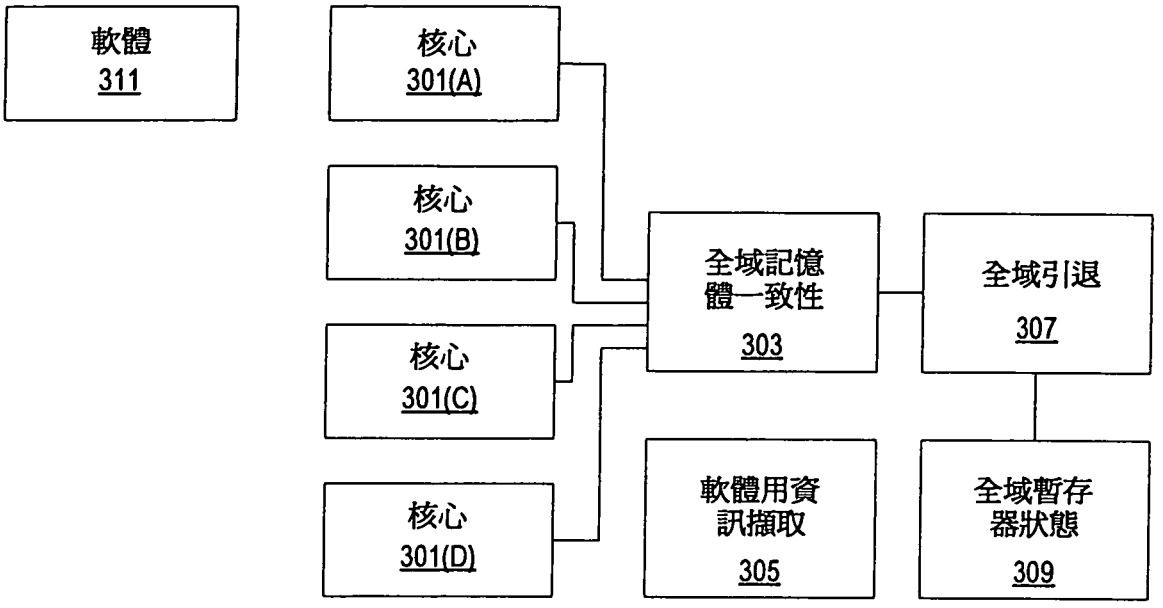
軟體
311
核心
301(A)
核心
301(B)
核心
301(C)
核心
301(D)
全域記憶體一致性
303
全域引退
307
軟體用資訊擷取
305
全域暫存器狀態
309

圖3

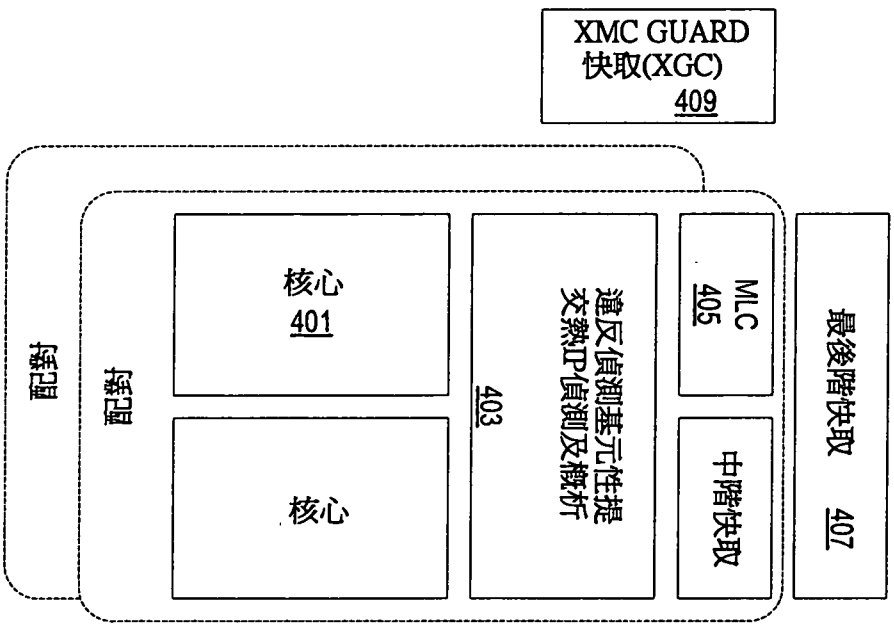
XMC GUARD
快取(XGC)
409
配對
配對
核心
401
核心
違反偵測基元性提
交熱IP偵測及概析
403
MLC
405
中階快取
最後階快取 407

圖4

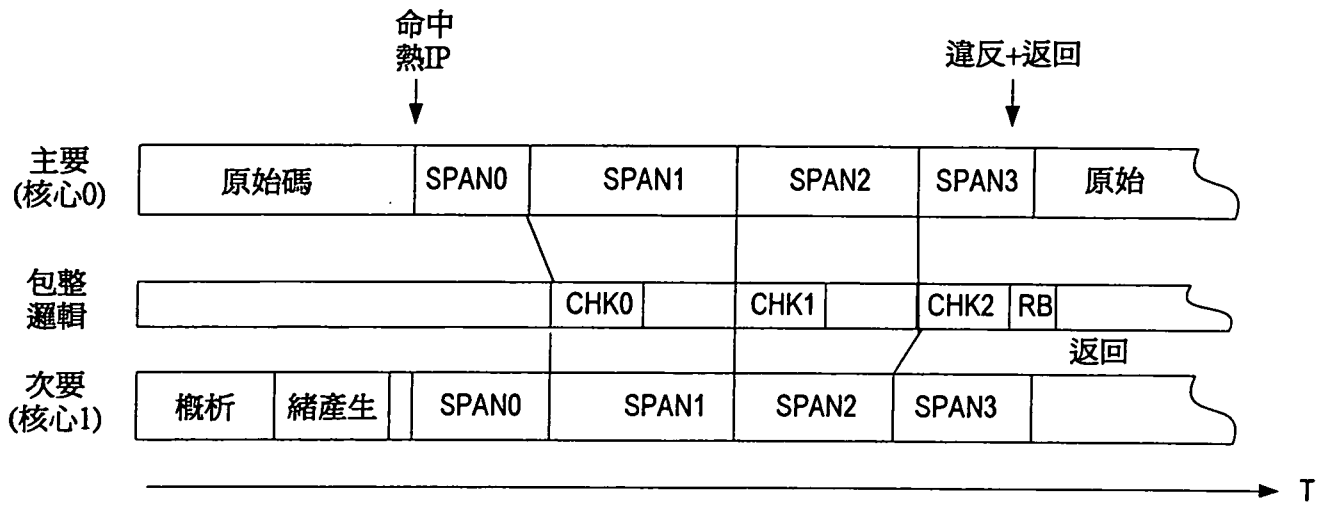
命中
熱IP
違反+返回
主要
(核心0)
原始碼
SPAN0
SPAN1
SPAN2
SPAN3
原始
包整
邏輯
CHK0
CHK1
CHK2
RB
返回
次要
(核心1)
概析
緒產生
SPAN0
SPAN1
SPAN2
SPAN3
T

圖5

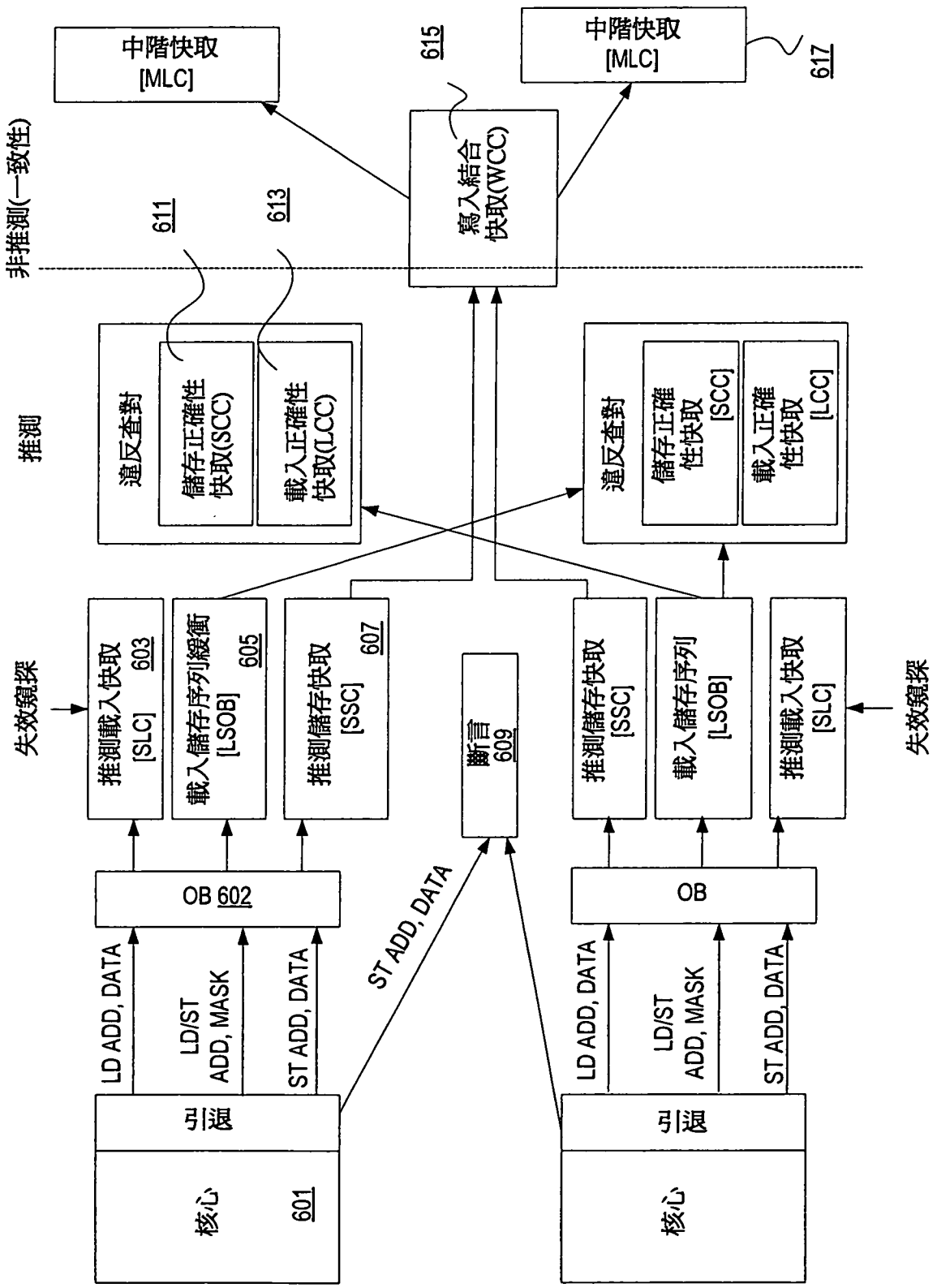
中階快取
[MLC]
615
中階快取
[MLC]
617
非推測(一致性)
611
613
寫入結合
快取(WCC)
推測
違反查對
儲存正確性
快取(SCC)
載入正確性
快取(LCC)
違反查對
儲存正確
性快取
[SCC]
載入正確
性快取
[LCC]
失效窺探
推測載入快取
[SLC] 603
載入儲存序列緩衝
[LSOB] 605
推測儲存快取
[SSC] 607
斷言
609
推測儲存快取
[SSC]
載入儲存序列
[LSOB]
推測載入快取
[SLC]
失效窺探
OB 602
OB
LD ADD, DATA
LD/ST
ADD, MASK
ST ADD, DATA
ST ADD, DATA
LD ADD, DATA
LD/ST
ADD, MASK
ST ADD, DATA
引擎
核心
601
引擎
核心

圖6

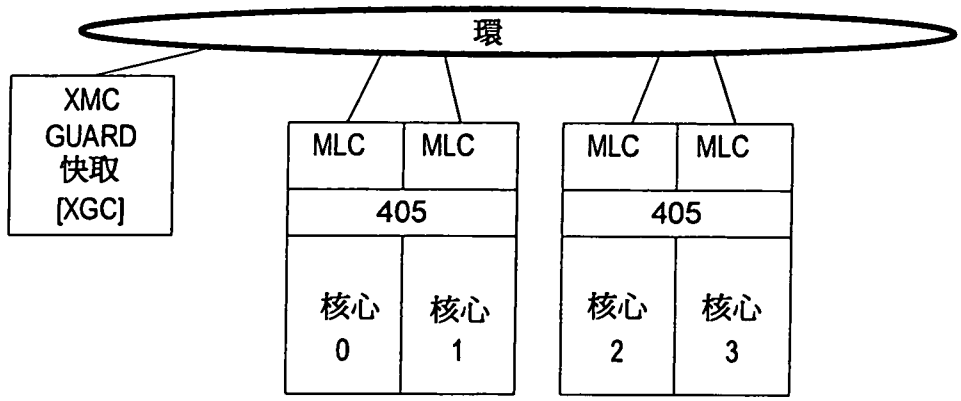

圖7

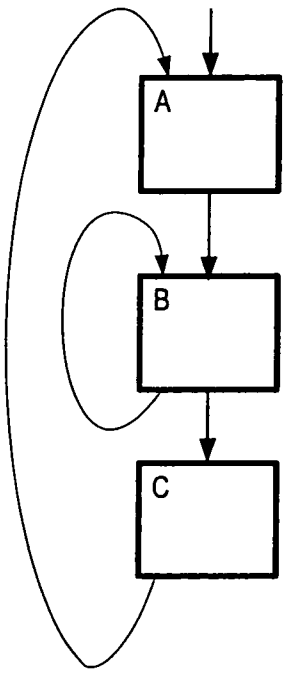

圖8

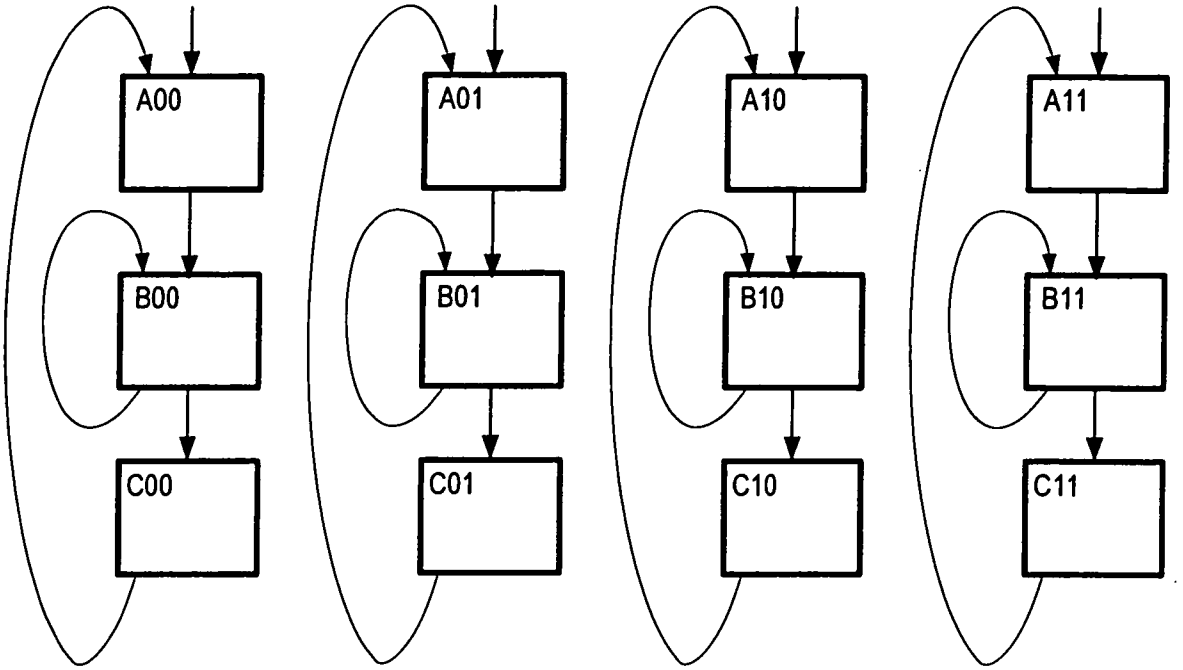

圖9

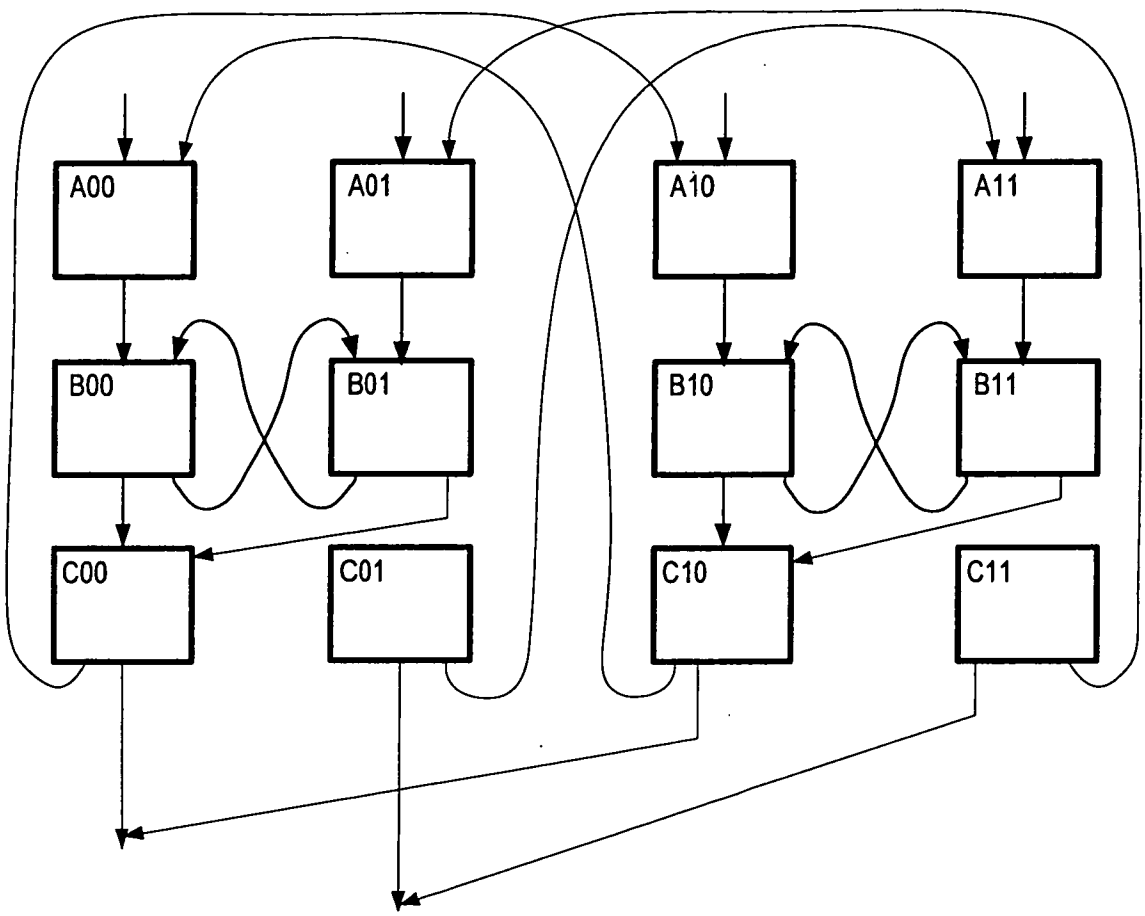

圖10

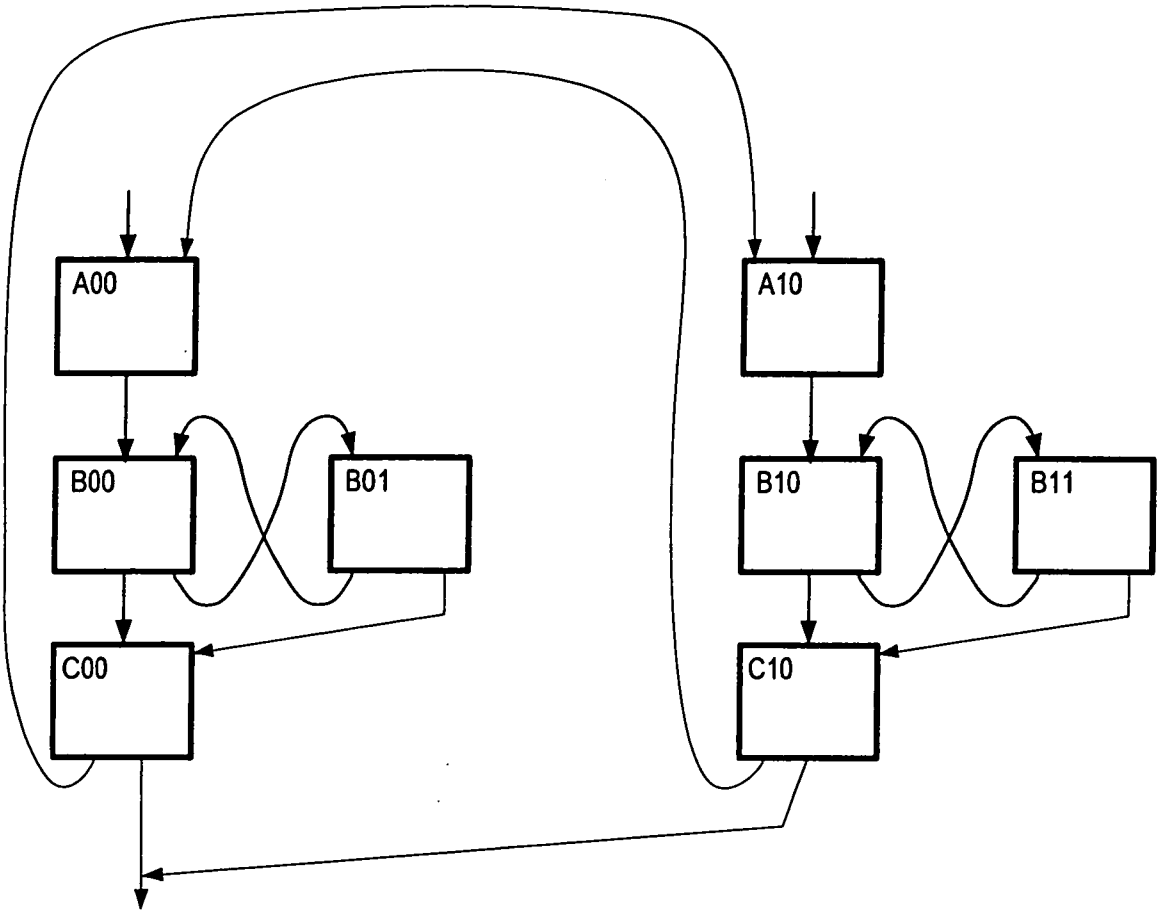

圖11

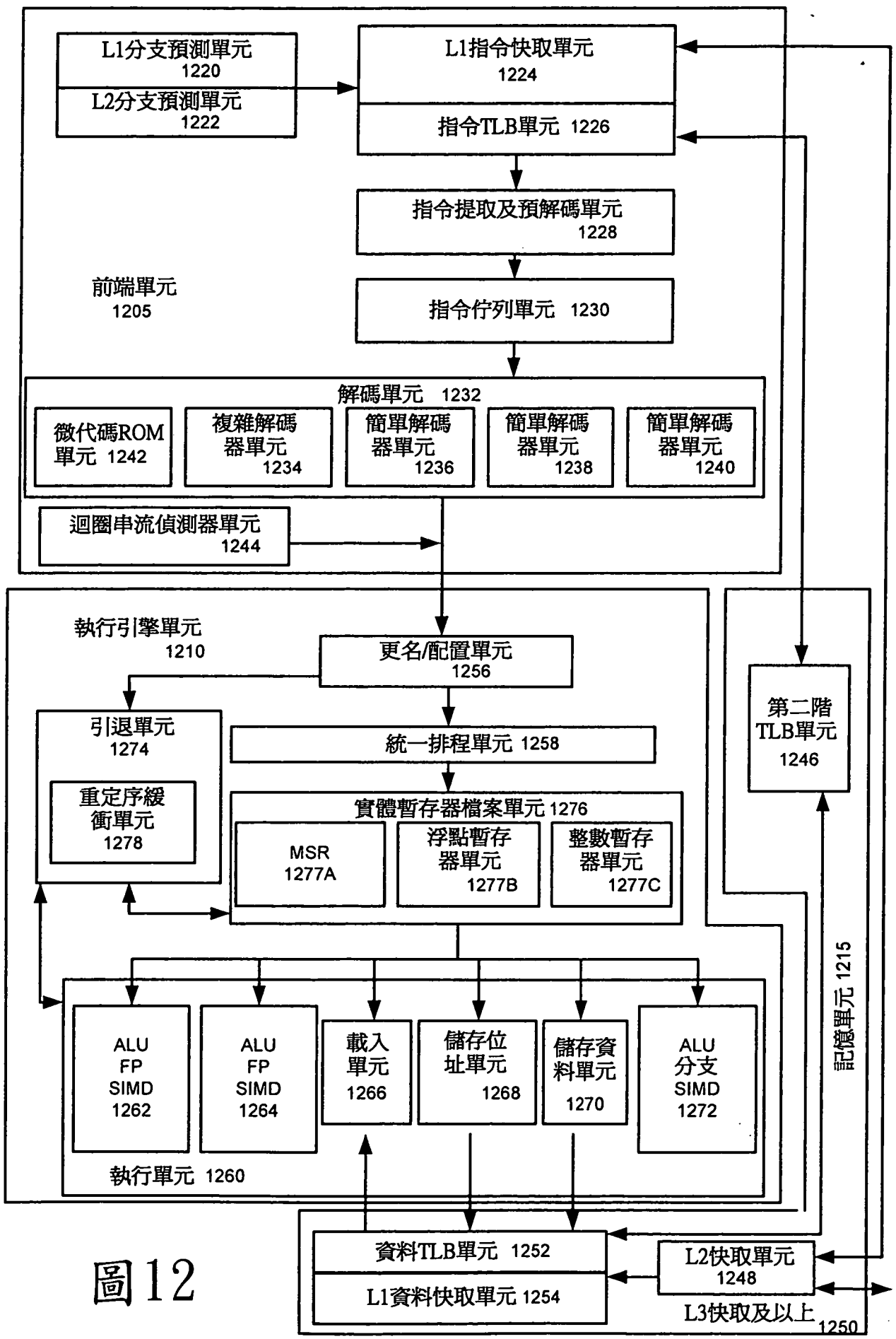
L1分支預測單元
1220
L2分支預測單元
1222
L1指令快取單元
1224
指令TLB單元 1226
指令提取及預解碼單元
1228
前端單元
1205
指令佇列單元 1230
解碼單元 1232
微代碼ROM
單元 1242
複雜解碼
器單元
1234
簡單解碼
器單元
1236
簡單解碼
器單元
1238
簡單解碼
器單元
1240
迴圈串流偵測器單元
1244
執行引擎單元
1210
更名/配置單元
1256
第二階
TLB單元
1246
引退單元
1274
重定序緩
衝單元
1278
統一排程單元 1258
實體暫存器檔案單元 1276
MSR
1277A
浮點暫存
器單元
1277B
整數暫存
器單元
1277C
ALU
FP
SIMD
1262
ALU
FP
SIMD
1264
載入
單元
1266
儲存位
址單元
1268
儲存資
料單元
1270
ALU
分支
SIMD
1272
執行單元 1260
記憶單元 1215
資料TLB單元 1252
L1資料快取單元 1254
L2快取單元
1248
L3快取及以上
1250

圖12

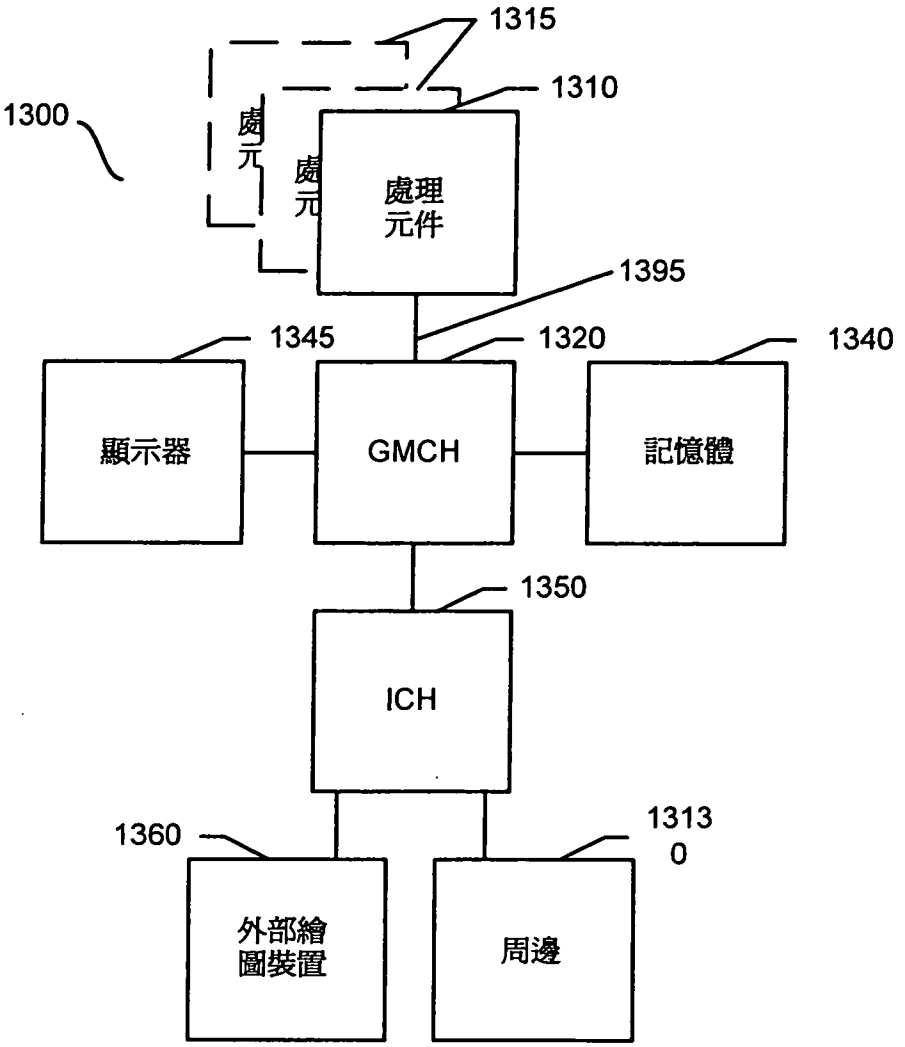
1315
1310
1300
處元
處元
處理元件
1395
1345
1320
1340
顯示器
GMCH
記憶體
1350
ICH
1360
1313 0
外部繪圖裝置
周邊

圖13

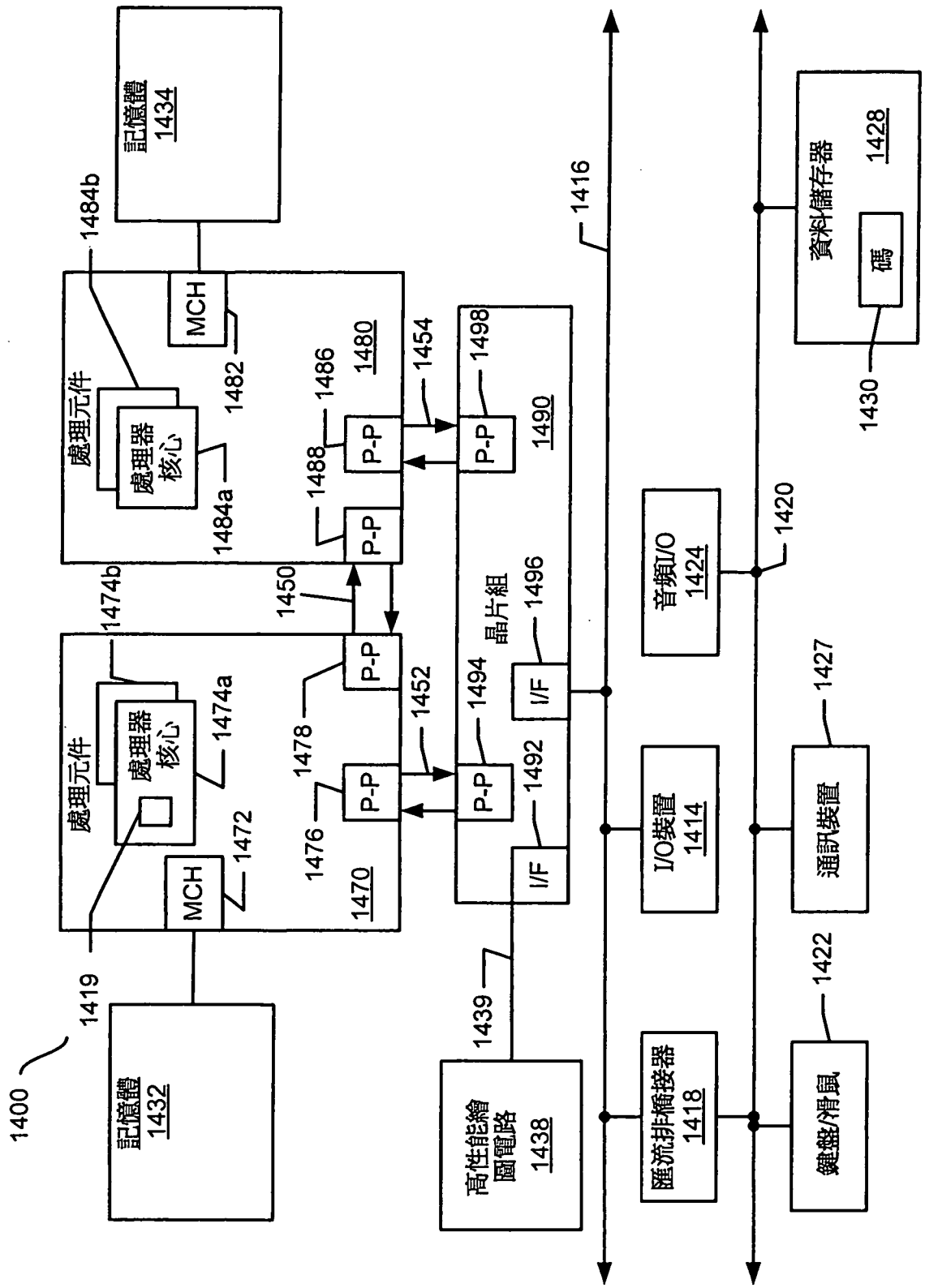
1400
1419
記憶體 1432
MCH
1472
處理元件
處理器核心
1474a
1474b
1476
1478
P-P
1452
1450
1454
1480
1482
1484a
1484b
1486
1488
記憶體 1434
MCH
處理元件
處理器核心
P-P
1498
1490
晶片組
1496
1494
1492
I/F
I/F
1439
高性能繪圖電路 1438
1416
音頻I/O 1424
1420
I/O裝置 1414
通訊裝置
1427
匯流排橋接器 1418
鍵盤/滑鼠
1422
資料儲存器
1428
碼
1430

圖14

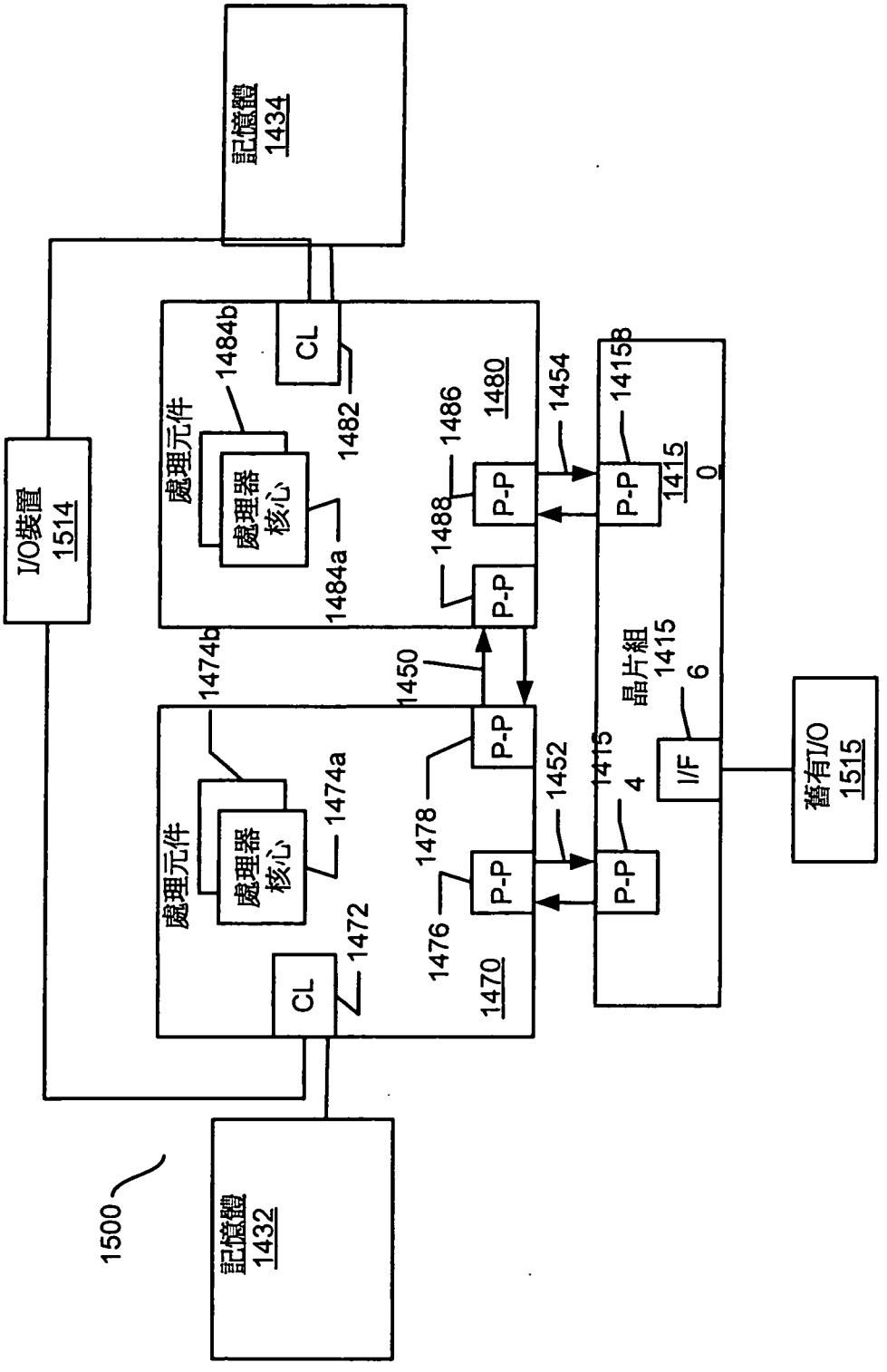
記憶體
1434
I/O裝置
1514
處理元件
處理器
核心
1484b
1484a
CL
1482
1480
1486
1488
P-P
1454
1450
1474b
1474a
1472
1478
1476
1470
1452
1415
4
1415
6
晶片組
1415
0
1415B
I/F
舊有I/O
1515
1500
記憶體
1432

圖15