目錄

1. 前言

近年來，隨著半導體與晶片設計的蓬勃發展，電子資訊應用不斷地從原本的個人電腦平台上整合到特定用途的嵌入式系統中，如目前已可透過手機播放音樂與收看數位電視節目，為了提供更多樣化的功能，嵌入式系統的處理效能也一直隨著需求向上提升，基於效能、耗電與成本等多重考量下，有越來越多的嵌入式系統晶片開始採用多核心 (multi-core) 架構之設計以提升效能並降低耗電與成本。然而，多核心處理器平台的硬體架構與軟體程式開發複雜度都較單一處理器平台高出許多，在系統晶片開發期間若沒有一套完整的工具輔助多核心嵌入式系統的硬體與軟體設計者分析、監控系統之效能與耗能狀況，多核心嵌入式系統晶片將可能會因為軟硬體協調不良或軟體設計不佳導致整個系統晶片效能低落、耗電嚴重，進而造成多核心嵌入式系統晶片的失敗。

2. 研究目的

本計畫的主要目的在於提出一套多核心嵌入式系統之效能分析與耗能評估方式，並將其實作成一套適用於多核心嵌入式系統的效能與耗能之監測與分析工具，以幫助多核心嵌入式系統開發者有效掌握多執行緒程式 (multi-thread program) 的效能與耗能。此一軟體工具由本計畫命名為「mProfiler」(Multicore Profiler)，其主要提供下列四項效能與耗能分析:

(1) **執行緒層級平行度 (Thread-Level Parallelism, TLP)**

在多核心系統上，軟體通常需由單行程 (process) 程式改寫成為多執行緒 (或多行程) 程式，才能讓多處理器核心同時處理多項工作以加速程式執行，為驗證軟體是否有提供足夠的執行緒讓多處理器核心平行執行，執行緒層級平行度的觀測相當重要。

(2) **負載平衡 (Load Balance)**

若要了解軟體是否已有效利用多核心處理器的硬體資源，除了檢視軟體提供的執行緒層級平行度之外，了解系統上的執行緒是否有被多核心處理器平行執行也是相當重要的問題，若是軟體提供了足夠的執行緒層級平行度，但作業系統卻沒有良好的負載平衡 (load balance) 機制將這些執行緒安排到各顆處理器核心上執行的話，將使得採用多核心處理器帶來的效能優勢大幅減少。

(3) **效能瓶頸 (Performance Bottleneck)**

為提供軟體設計者更進一步的建議，本計畫將分析一個多執行緒程式中每個執行緒的生命週期，讓軟體設計者了解每個執行緒的效能瓶頸，進而建議設計者針對演算法、執行緒間的資料同步負擔 (synchronization overhead) 或是平行度做改善。

(4) **處理器耗能評估 (Energy Estimation)**

上述三點主要是針對效能的部份做分析，但在多核心嵌入式系統中，高效能並非唯一的考量，多核心的架構雖然可以有效增加程式的效能，但增加的核心數量同時也會為處理器帶來更大的功率消耗 (power consumption)，因而讓多核心處理器可能需要花費比單核心處理器更多的能量 (energy) 才能完成一件相同的工作，使得系統的能量效率 (energy efficiency) 不彰，能量效率低落表示系統上的能量並沒有得到妥善運用，如此可能會增加電費成本或是縮短手持裝置的電池使用時間，因此，mProfiler 亦會針對處理器的耗能做評估，以作為多核心嵌入式系統開發者改善軟硬體設計的參考依據。

本計畫提出之 mProfiler 將透過上述四點效能與耗能分析，協助多核心嵌入式系統設計者在系統開發初期即有能力發現並了解效能或是耗能上的問題，以對多核心嵌入式系統的軟體和硬體做最佳化的設計。

3. 相關研究

相關的議題中，國內的研究計畫多以提出新式設計以改進嵌入式系統晶片效能[1] [2] [3] 或減少系統耗能[4] [5] [6] [7] 為主，較少探討如何評估與分析系統效能與耗能；而在國外的相關研究中主要都是對單核心的硬體平台做效能和耗能的監測與分析，以多核心系統晶片為基礎研究平台的系統效能與耗能分析則較為少見，其中，柏克萊大學目前正進行的 RAMP 計畫是一項關於多核心系統的平台建置計畫[8] ，然而 RAMP 計畫並不強調嵌入式的應用，因此效能與耗能監測等工具的發展相對著墨較少。有關如何在多核心嵌入式系統上建構與效能和耗能監測相關的軟硬體元件，以及配套的分析工具，目前在國內外的研究中是比較少見的。

多核心嵌入式系統晶片之效能評估上，通常以處理器執行該工作所花的時間為準[9] ，而在耗能評估上，現有的方式中主要可分為基於監控 (monitor-based) 與基於模型

(model-based) 兩種耗能評估方式，其中基於監控的耗能評估方式[10] [11] 在測量耗能時，會先把主電源接到一個數位電表 (digital multimeter) 中，再經由數位電表提供待測系統所需電能，之後當待測系統開始執行時，便可以藉由數位電表中的電流值即時監控取樣時間點上待測系統的耗能狀況，接著再針對這些取樣點逐一紀錄系統狀況與耗能狀況，便可於測量結束之後，對待測系統執行時所收集到的狀態與能量數據進行統計及分析，以評估整體的耗能狀況。此類型的耗能評估方式快速且準確度高、誤差度小，不過對使用者而言卻不甚便利，因為此耗能評估方式必須與數位電表搭配使用，若使用者沒有符合此耗能評估方式所需的數位電表，整個評估機制將無法順利啟動。

而基於模型的耗能評估方式會先將硬體的耗電程度量測好，並利用軟體的方式來評估系統耗能，以避免缺乏硬體資源的問題產生，讓使用者無須外接任何硬體設備即可評估系統的耗能狀況，此評估方式依精確度又可進一步分為週期程度 (cycle-level) 精確度、指令程度 (instruction-level) 精確度與高階 (higher level) 精確度三種。其中精確度最高之週期程度精確度的評估方式中[12] [13] [14] ，主要是利用一個軟體模擬器，模擬待測系統中每個時脈週期中所有硬體 (如：處理器、記憶體、匯流排等) 的行為與其對應的消耗能量，以達到評估系統耗能的目的，但也由於是對整個硬體架構做模擬，因此若要實作此耗能評估機制，必須要相當了解待測系統的硬體架構。而指令程度精確度的評估方式中[15] [16] ，會事先量測好在待測系統上執行每個指令，以及指令間交互影響 (inter-instruction effect)、快取誤失 (cache misses) 和管線延遲 (pipeline stalls) 等所需的耗電量，最後再透過指令集模擬器 (instruction set simulators) 取得程式執行時每個指令出現的次數以及發生前述事件的數量，以統計分析整體系統耗能，使用此方法的主要缺點是若待測系統採用複雜指令集 (Complex Instruction Set Computing, CISC) 架構，則系統中將會有相當可觀的指令數量需要事先量測其耗電量。在前述兩種精確度的量測方式中，皆有著效率不佳以及難以實做的問題存在，因此後來便有高階精確度的評估方式被提出來，[17] 提出在某些採用精簡指令集 (Reduced Instruction Set Computing, RISC) 架構的處理器 (如: ARM) 中，每個指令間所耗費的電能差異度相當低，且不同程式間的耗能差異度約為 8% 左右，因此在對該類型的系統平台做耗能分析時，並不需要很詳盡地測量每個指令的耗電量以及分析所有指令間的交互影響行為，而是可以利用一個較簡單的耗能模型去評估此系統的耗能狀況即可。除了前述

的簡化方法之外，另外還有軟體巨集模型 (Software macromodeling) [18] 和函式程度 (function level) 精確度[19] 的評估方式,與針對作業系統之系統呼叫 (system calls) [20] [21] [22] 等，利用事先量測好程式中部份程式區塊或函式之耗電量的作法相繼被提出來，以加快評估系統耗能的速度。

然而，以上方式皆是針對單核心系統平台所提出來的效能與耗能評估方式，無法完全套用在行為更為複雜的多核心系統平台，因此本計畫將提出一套適用多核心嵌入式系統的效能與耗能之監測與分析工具 (mProfiler)，讓多核心嵌入式系統晶片與軟體設計者在開發初期即可有效地掌握效能與耗能的特性，除了可以避免預期外的耗能狀況發生之外，還可以進一步針對效能與耗能部份做最佳化的處理，提高多核心嵌入式系統晶片之整體效能並降低耗能。

4. 研究方法

本計畫主要是提出一套多核心嵌入式系統之效能分析與耗能評估方式，並基於一個現有的多核心嵌入式系統開發平台 (ARM11 MPCore 開發平台 [23] [24] [25])，實做出一套適用多核心嵌入式系統的效能與耗能之監測與分析工具 (mProfiler)，就開發環境上來說，本計畫使用的作業系統核心是 Linux kernel 2.6.19，開發工具是 GNU Toolchain for ARM Processors GCC 4.2.1 與 Glibc 2.5。本計畫開發之工具將提供多核心嵌入式系統設計者效能分析 (Performance Analysis) 和耗能評估 (Energy Estimation) 兩大類資訊，以下章節將先說明 mProfiler 的軟體架構 (software architecture)，再分別描述效能分析與耗能評估的設計與實做方式。

4.1. 軟體架構 (Software Architecture)

本計畫提出之 mProfiler 的軟體架構圖如圖 1 所示，圖中依據硬體平台 (ARM11 MPCore Platform)、作業系統 (GNU/Linux) 和使用者程式 (User Space) 分成三層，並以白色方塊標示出既有之軟體或是硬體元件、藍色方塊標示出本計畫加入或修改的程式碼，而橘色方塊則是表示本計畫蒐集的資料，包含時間 (time) 與功率 (power) 的相關資訊。

為了讓使用者方便使用，mProfiler 主要是以使用者程式的方式呈現 (如圖 1 之 "mProfiler" 區塊所示)，多核心嵌入式系統開發者可以透過 mProfiler 創造 (fork) 出目標軟體 (如: benchmark program) 的行程 (process)，此時本計畫擴充之作業系統核心 (kernel extension) 便會開始動態蒐集目標軟體的時間資訊並儲存於作業系統核心中，在目標程式執行期間，mProfiler 也會週期性地透過系統呼叫 (system call) 的方式讀取並蒐集硬體平台上 ADC (Analog to Digital Converter) 提供的功率資訊，且將蒐集到的功率資訊儲存於 mProfiler 中。當目標軟體結束執行 (exit) 之後，mProfiler 也會跟著結束執行，並將蒐集到之時間與功率資訊做整理歸納之後統一回報給使用者。另外，為了解軟體的效能與耗能對核心數量變化的擴充性 (scalability) 優劣，mProfiler 也可以透過 ARM11 MPCore 處理器的驅動程式 (device driver) 將部份處理器核心切換至 WFI (Wait for Interrupt) 模式，以達到變動系統處理器核心數量的目的。

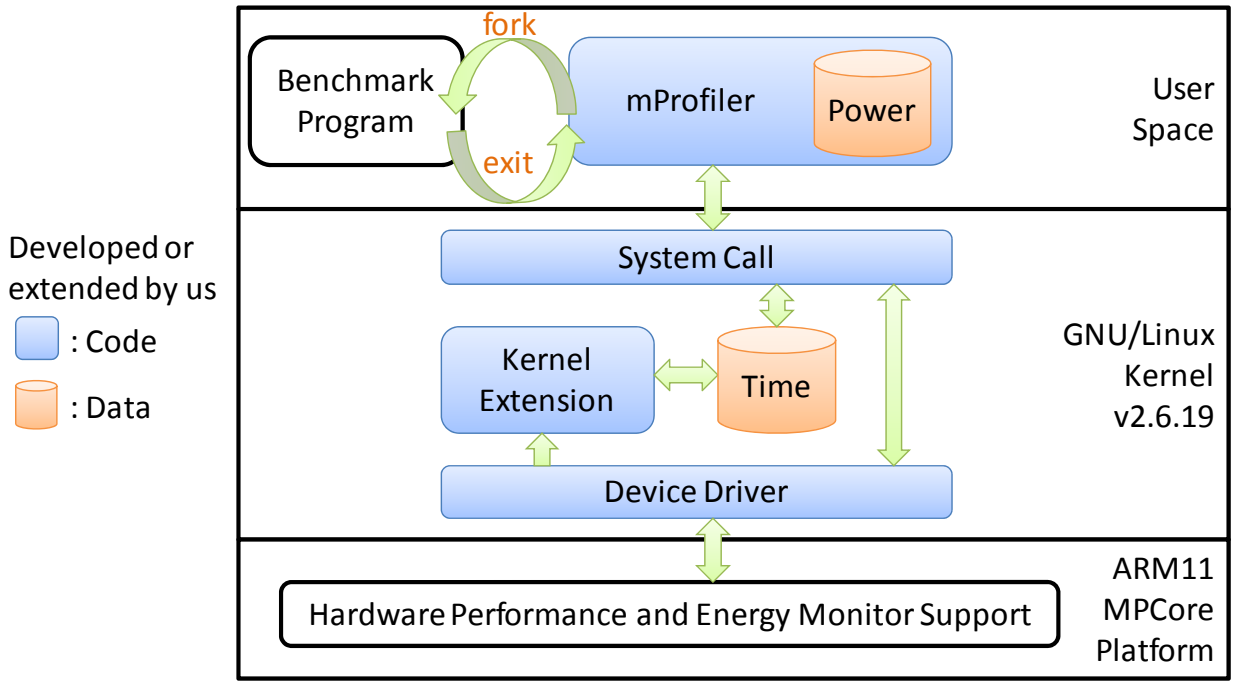


圖 1　mProfiler 軟體架構圖

4.2. 效能分析 (Performance Analysis)

就效能分析上來說，本計畫提出之工具 (mProfiler) 主要針對軟體的執行緒層級平行度 (Thread-Level Parallelism, TLP)、負載平衡 (Load Balance) 與軟體效能瓶頸 (Performance Bottleneck) 等三大項目做評估，以下將個別說明之。

4.2.1. 執行緒層級平行度 (Thread-Level Parallelism, TLP)

在多核心系統上，軟體通常需由單執行緒 (或單行程) 程式改寫成為多執行緒 (或多行程) 程式，才能提供足夠的工作量讓數個處理器核心同時處理，進而縮短程式的生命週期 (life time)，為驗證一個多執行緒軟體是否在執行期間總是提供足夠的工作讓硬體執行，mProfiler 會紀錄多執行緒程式中，每個執行緒被創造出來的時間點以及它們個別的生命週期長度，以讓多執行緒軟體開發者了解程式中每個執行緒是否有如預期般地被創造與執行，詳細的執行結果請參考 5.1 的說明。

執行緒被創造出來的時間點是在 Linux kernel 中 `kernel/fork.c` 檔案裡的 `do_fork()` 函式[26]，而執行緒結束的時間點是在 `kernel/exit.c` 檔案裡的 `do_exit()` 函式[26]，因此 mProfiler 便分別在上述兩個檔案中加入程式碼，把每個執行緒被創造出來的時間與從被創造出來到執行結束的生命週期時間，紀錄到本計畫在每個執行緒的 `task_struct` 裡加入的資料欄位內。

4.2.2. 負載平衡 (Load Balance)

若要了解軟體是否已有效利用多核心處理器的硬體資源，除了檢視軟體提供的執行緒層級平行度之外，還必須要確認系統上的執行緒是否有被多核心處理器平行執行，若是軟體提供了足夠的執行緒層級平行度，但這些執行緒卻只在少數幾顆處理器核心上執行的話，則多核心處理器的硬體資源仍不算得到有效利用。為此，mProfiler 將辨識出每個執行緒的執行時間 (run time) 花費在各顆核心上的比例，讓多核心嵌入式系統的開發者能夠了解系統上的執行緒是否有妥善利用多核心處理器提供的硬體資源，以避免工作分配不均勻而導致能量效率不佳的狀況發生。

在 Linux kernel 中，可以透過呼叫 `smp_processor_id()` 巨集 (macro) [26] 得知執行目前程式碼的處理器核心編號，因此只要在 `kernel/sched.c` 的 `schedule()` 函數內，將統計到的執行時間透過處理器核心編號分類統計，便可取得執行時間花費在各處理器核心上的比例。

4.2.3. 軟體效能瓶頸 (Performance Bottleneck)

為提供軟體設計者更進一步的建議，本計畫把一個多執行緒程式中每個執行緒的生命週期分成執行時間 (run time)、等待時間 (block time) 和其他時間 (other time)，讓軟體設計者了解每個執行緒的效能瓶頸，進而針對演算法、執行緒間的資料同步負擔 (synchronization overhead) 或是平行度做改善，各類時間的意義與取得方法說明如下：

(1) **生命週期 (life time)**

執行緒從被創造出來到執行結束的時間長度。此時間可被用來評估執行緒層級平行度，詳細說明請參考 4.2.1。mProfiler 從 `kernel/fork.c` 檔案裡的 `do_fork()` 函式替新執行緒建立完 `task_struct` 之後開始計時，直到執行緒執行 `kernel/exit.c` 檔案裡的 `do_exit()` 函式之後才停止計時。

(2) **執行時間 (run time)**

執行緒被處理器執行的時間總和。mProfiler 除了提供執行時間的累計值之外，另外還提供執行時間在使用者模式 (user mode) 和核心模式 (kernel mode) 下的比例，若是使用者模式佔去大部分的執行時間，表示使用者程式的演算法需要做進一步的改善；反之，若是核心模式佔了大部分的執行時間，則表示使用者程式使用了過多的系統呼叫

(system call) 或是系統呼叫的程式碼需要改善。執行時間的累計值可透過修改 Linux kernel 的 `kernel/sched.c` 檔之 `schedule()` 函數取得，即讓執行時間在目標執行緒被內容切換 (context switch) 至執行狀態 (running state) 時開始計時，直到該執行緒從執行狀態被切換回預備狀態 (ready state) 後才停止計時並做時間累計。而使用者模式 (user mode) 和核心模式 (kernel mode) 的時間比例則是利用 kernel 在執行緒之 `task_struct` 內維護統計的 `utime` 和 `stime` 兩個欄位作為參考。

(3) **等待時間 (block time)**

執行緒間因資料同步 (synchronization) 而需等待的時間總和。當一個程式從原本的單執行緒程式改寫成為多執行緒程式後，程式的執行緒間通常需要透過共享資料 (shared data) 來相互溝通以便進行工作分配等事宜，為確保共享資料的正確性，執行緒會利用適當的同步機制 (如: mutex, semaphore) 來保護共享資料。mProfiler 藉由提供同步機制帶來的等待時間長短，讓軟體開發者能夠了解各執行緒間溝通機制的優劣，進而調整資料同步的時間點或是減少不必要的共享資料量，以增加執行緒的效能。本計畫目前是針對 POSIX Threads 函式庫使用到之 futex 同步元件進行等待時間量測，因此在 `kernel/futex.c` 檔內的 `futex_wait()` 函式內開始計時，並在 `futex_wake()` 函式中結束計時並累計時間。

(4) **其它時間 (other time)**

此時間是由執行緒的生命週期扣除執行時間和等待時間得來，若此時間很長，表示系統中可能有過多的工作，讓處理器核心忙於處理系統上的其他執行緒而使得目標執行緒的生命週期變長，這個訊息告訴多執行緒程式的開發者，與其改良執行緒的演算法，不如減少系統中之執行緒數量，可能可以更有效地縮短多執行緒程式的生命週期。

4.3. 耗能評估 (Energy Estimation)

以耗能評估而言，mProfiler 主要以 ARM11 MPCore 之 Core Tile 上的 ADC (Analog to Digital Converter) 作為得到功率 (power) 資訊的參考來源，ARM11 MPCore 之 Core Tile 上具有一個 ADC 裝置[24] ，如圖 2 所示，該 ADC 裝置會持續地偵測流經處理器和 PLL (Phase-Locked Loop) 的電壓值和電流值，並將這些數值轉換成數位資料之後，存入 ARM11 MPCore 評估板 (Emulation Baseboard) 上的暫存器 (SYS_SET_VOLTAGE*x*) 中，之後軟體

便可以透過讀取這些暫存器並做下列運算，而得到處理器和 PLL (Phase-Locked Loop) 的電壓值和電流值，進而計算它們的功率和耗能。

處理器 (包含 4 顆核心的總耗電):

電壓值 = (SYS_VOLTAGE0[19:8] * 610) uV

電流值 = ((SYS_VOLTAGE2[19:8] * 610) / 2.5) uA

PLL:

電壓值 = (SYS_VOLTAGE1[19:8] * 610) uV

電流值 = ((SYS_VOLTAGE3[19:8] * 610) / 50) uA

關於暫存器 (SYS_SET_VOLTAGE*x*) 的實體位址與格式可參考 ARM11 MPCore 之 Core Tile 的說明文件 [24] ，在得知這些暫存器的實體位址之後，便可利用系統呼叫的方式，讓使用者程式得以週期性地取得處理器和 PLL 的電壓和電流值。目前，mProfiler 會以每 10ms 取樣一次的頻率，取得處理器 (包含四顆核心) 和 PLL 的電壓值和電流值，並在目標程式結束執行之後，顯示出處理器和 PLL 的平均電壓 (V_{avg})、平均電流 (I_{avg}) 與總耗能 (E)，其個別計算方式如下所示:

$$V_{avg} = \frac{\sum_{i=1}^{N_s} V_i}{N_s}, \quad I_{avg} = \frac{\sum_{i=1}^{N_s} I_i}{N_s}, \quad E = T_s \cdot \sum_{i=1}^{N_s} (V_i \cdot I_i)$$

其中，N_s 是取樣點的數量、V_i 是第 i 次的電壓取樣值、I_i: 是第 i 次的電流取樣值、T_s 是取樣時間間隔長度 (此處是 10 ms)。

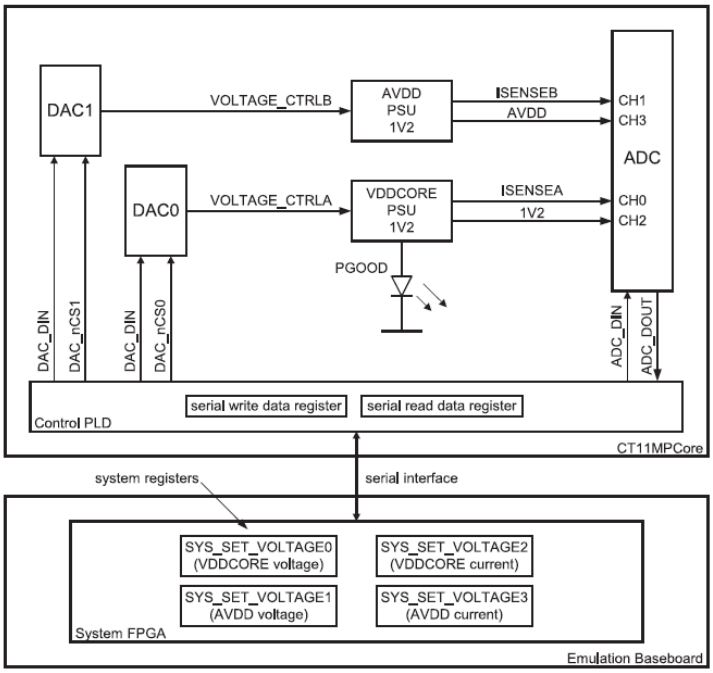


圖 2 Voltage control and voltage and current monitoring [24]

5. 結果與討論

本計畫提出之適用多核心嵌入式系統的效能與耗能之監測與分析工具 (mProfiler)，主要提供 (1) 執行緒層級平行度、(2) 負載平衡、(3) 軟體效能瓶頸、以及 (4) 處理器耗能評估等共四大項資訊供多核心嵌入式系統開發者參考，本報告的第 4 章說明了這些資訊的意義與取得方式，而此章以 ALPBench benchmark suite [27] 中的 MPEG-2 decoder 為例，展示並說明透過 mProfiler 可得到的效能分析與耗能評估結果。

5.1. 效能分析結果

就效能分析部份，首先說明的是如何透過 mProfiler 檢視軟體的執行緒層級平行度，該工具提供之相關資訊如圖 3 所示，圖中的水平軸為由 MPEG-2 decoder 產生出來之所有執行緒的編號，而垂直軸為時間，圖中藍色長條圖的長度描繪出各執行緒的生命週期長度，而所有執行緒的起始執行時間，將會從相對於主執行緒 (Task ID = 324) 之起始執行時間的位置上開始描繪。透過此資訊，多核心嵌入式系統開發者可以了解在任一時間點之軟體執行緒層級平行度，舉例來說，由圖 3 可以看出，此 MPEG-2 decoder 在 0ms ~ 100 ms 之間與 260 ms ~ 360 ms 之間，都只透過單一執行緒在處理工作，此程式只有在 100 ms ~ 260 ms 之間 (不到程式生命週期的一半時間) 有維持 4 ~ 5 個執行緒在系統上平行執行，對一個具有四核心處理器的實驗平台來說，這個實驗結果表示此目標程式的執行緒層級平行度仍有待改進。

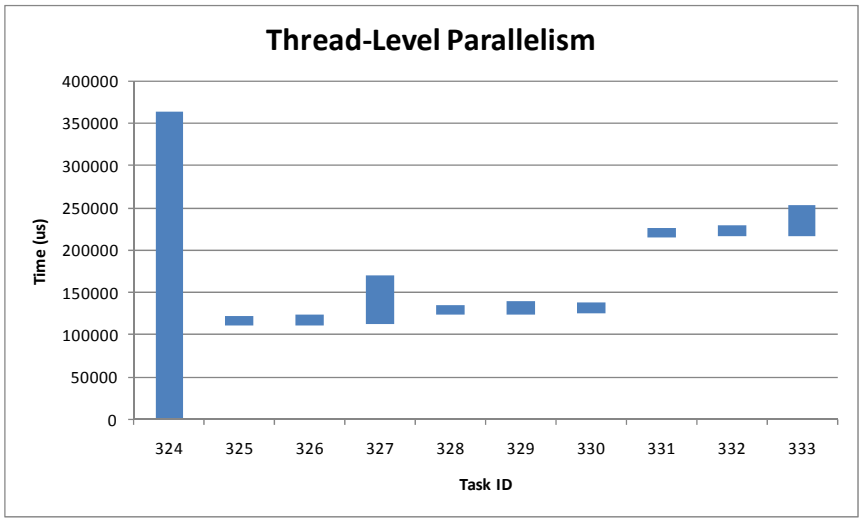
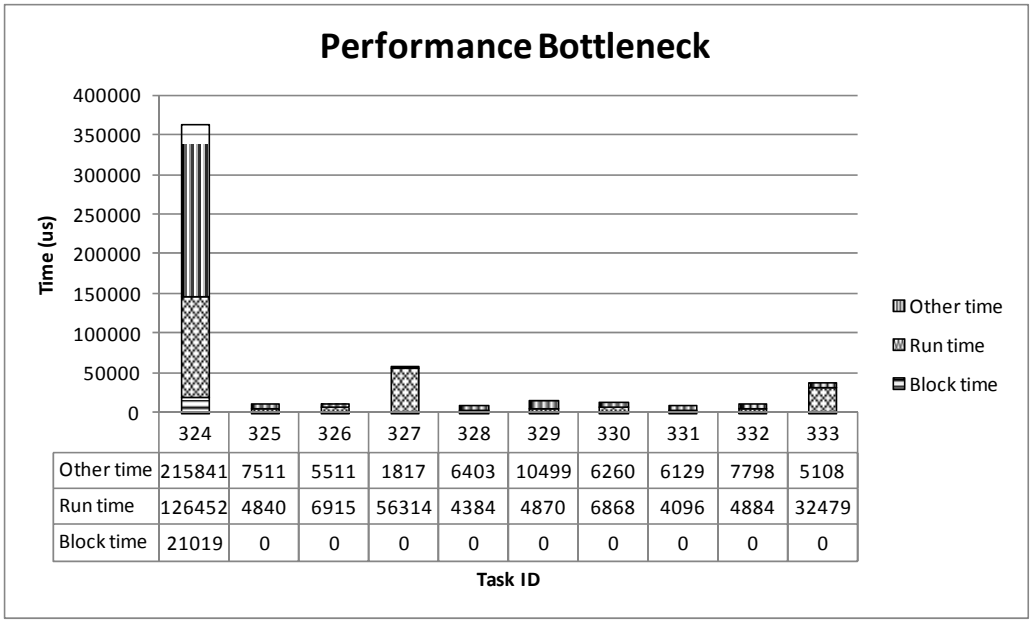


圖 3　執行緒層級平行度之檢視圖

為輔助多核心嵌入式系統開發者更了解軟體效能瓶頸，因此 mProfiler 提供生命週期的時間分析，如圖 4 所示，圖中水平軸和垂直軸與圖 3 相同，每個長條圖由執行時間 (run time)、等待時間 (block time) 和其它時間 (other time) 等三種時間堆疊而成，這三種時間加起來的總長度即為執行緒的生命週期長度。從圖 4 顯示的執行時間長度可以發現，MPEG-2 decoder 程式中，每個執行緒的工作分配量並不均勻，即編號 324, 327 和 333 這三個執行緒共被分配到八成以上的工作，而其他七個執行緒則僅被分配到剩下不到兩成的工作量，如此工作分配不勻的狀況將使得多執行緒程式無法有效利用多核心處理器之硬體優勢，也會導致程式的效能擴充性 (scalability) 不佳，因此多核心嵌入式系統開發者應盡可能平均分配工作量，讓各個處理器核心能一起進行與完成工作，以減少較早完成工作之處理器核心等待其他核心處理較繁重的工作所花費的時間和能量。

另外，光從生命週期長度來看，會以為編號 324 的主執行緒被分配到相當多的工作量，但藉由圖 4 的軟體效能瓶頸分析可以觀察到，佔據主執行緒一半以上時間的是其它時間而非執行時間或是等待時間，這表示主執行緒花了大部分的時間在等待處理器核心的使用權，造成這個問題的原因，可以從圖 3 提供之執行緒層級平行度得到部份資訊，即當系統中的執行緒數量超過處理器核心數量時，執行緒之間便需要輪流使用處理器核心，進而讓執行緒的其他時間的長度增加，如圖 3 在大約 130 ms 的時候，系統中有 5 個執行緒，超過硬體 4 顆處理器核心的數量，如此便會造成執行緒的其他時間變長。



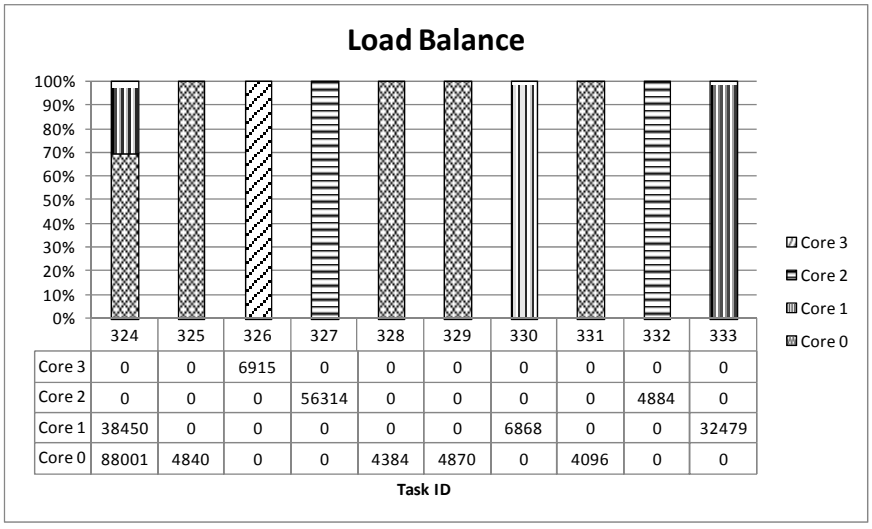
	324	325	326	327	328	329	330	331	332	333
Other time	215841	7511	5511	1817	6403	10499	6260	6129	7798	5108
Run time	126452	4840	6915	56314	4384	4870	6868	4096	4884	32479
Block time	21019	0	0	0	0	0	0	0	0	0

圖 4　軟體效能瓶頸之檢視圖

然而，僅參考圖 3 仍無法確定軟體提供的執行緒是在哪顆處理器核心上執行，亦無法確定執行緒的其他時間究竟是受到哪些執行緒的影響而增長，為此，mProfiler 進一步針對各個執行緒在各顆處理器核心上的執行時間做分類統計，如圖 5 所示，圖中的水平軸為執行緒編號，垂直軸為執行時間百分比，四種不同顏色的長條圖分別表示四顆不同的處理器核心所佔的執行時間百分比。藉由圖 5 提供的資訊，多核心嵌入式系統開發者可以得知，主執行緒有 70% 的執行時間是在編號 0 的處理器核心上執行、另有 30% 是在編號 1 的處理器核心上執行，因此主執行緒很有可能是受到其他同樣在編號 0 和編號 1 上執行的執行緒 (如: 編號 325, 328, 329, 330, 331, 333) 的影響才使得其他時間變長。

一個因為數個執行緒被安排給相同的處理器核心處理因而導致其他時間變長的明顯例子，如編號 328 和 329 這兩個執行緒所表現出來的行為所示，從圖 3 中可以觀察出這兩個執行緒幾乎是同時被創造出來，透過圖 5 可以看出，這兩個執行緒同被安排在編號 0 的處理器上執行，而從圖 4 中可以發現編號 328 和 329 這兩個執行緒被分配到的工作量其實差不多 (因為執行時間相近)，但因為編號 329 執行緒花了較多其他時間在等待編號 328 執行緒使用處理器核心，才使得編號 329 執行緒的生命週期較編號 328 執行緒的生命週期還要長。

以上便是本計畫提出之 mProfiler 提供的效能分析資訊，藉此讓多核心嵌入式系統開發者了解軟體的執行緒層級平行度、負載平衡以及軟體的效能瓶頸，之後多核心嵌入式系統開發者便可針對需求進行軟硬體設計的改良 (如: 平均分配軟體工作量、決定系統上之處理器核心數量等)，以增加系統效能或是減少耗能。



	324	325	326	327	328	329	330	331	332	333
Core 3	0	0	6915	0	0	0	0	0	0	0
Core 2	0	0	0	56314	0	0	0	0	4884	0
Core 1	38450	0	0	0	0	0	6868	0	0	32479
Core 0	88001	4840	0	0	4384	4870	0	4096	0	0

圖 5　負載平衡

5.2. 耗能評估結果

本計畫提出之 mProfiler 所測得的耗能結果如表 1 和表 2 所示，其分別列出 ARM11 MPCore 處理器 (包含 4 顆 MP11 處理器核心) 與 PLL 的平均電壓、平均電流與總耗能值，這三種數值的計算方式請參考章節 4.3 的說明。以平均電壓值而言，處理器和 PLL 都約莫是 1.2 V，而平均電流的部份處理器是 470 mA 左右，而 PLL 的則是 190 mA 左右。

本計畫在實驗過程中，也有嘗試關閉部份 ARM11 MPCore 處理器中的核心，然而，量測出來的耗能結果並不如預期，表 3 列出在系統閒置的情況下，變動處理器核心數量取得之平均電壓值與平均電流值，從表中可以看出無論系統中目前有幾顆處理器核心在運作，平均電壓都維持在 1.2 V 左右，而平均電流的部份，原本預期減少一顆核心運作能夠減少大約四分之一左右的電流值 (即 110 mA 左右)，但從表 3 中可以看出每關閉一顆核心實際上只少了 (1 mA ~ 2 mA 左右)，目前推估這個狀況發生的原因是因為本計畫採用的 ARM11 MPCore 發展板上並沒有 power controller 硬體元件可以將輸入單一處理器核心的電源移除，會這麼認為是因為雖然 ARM 提供的文件[25] 中有提到每顆 MP11 處理器核心都有四種功率模式 (power mode) (請參考圖 6)，且每顆處理器核心都具有獨立的 voltage domain，但即使 mProfiler 依據文件[25] 設定 SCU CPU Status Register，都無法成功把處理器核心從平常模式 (normal mode) 切換至斷電模式 (power-off mode) 或是休眠模式 (dormant mode) 並降低應有的電流值，只能把處理器核心切換至 WFI (wait for interrupt) 模式，並降低如表 3 所示的電流幅度。由於無法關閉處理器核心的電源，因此我們目前無法提供調整處理器核心數量並執行同一件工作所需耗電量的實驗結果。

表 1　處理器之平均電壓、平均電流與總耗能

Processor (4-Core)		
Average Voltage (uV)	Average Current (uA)	Total Energy (uJ)
1233549.394	469692.606	191203.41

表 2　PLL 之平均電壓、平均電流與總耗能

PLL		
Average Voltage (uV)	Average Current (uA)	Total Energy (uJ)
1242477.576	19039.764	7806.47

表 3　變動處理器核心數量取得之平均電壓值與平均電流值

The number of cores	Average voltage (uV)	Average current (uA)
4	1230193.100	440731.588
3	1230057.070	439127.044
2	1230116.850	437874.348
1	1229793.550	436677.284

Mode	MP11 CPU logic	RAM arrays	Wake-up mechanism
Run Mode	Powered-up Everything clocked	Powered-up	N/A
WFI/WFE	Powered-up Only wake-up logic clocked	Powered-up	Wake-up on interrupts (external or timer/WD). L1 memory system only wake-up in case of SCU coherency request.
Dormant	Powered-off	Retention state/voltage	External wake-up event to power controller.
Powered- off	Powered-off	Powered-off	External wake-up event to power controller.

圖 6　MP11 CPU power modes [25]

目前 mProfiler 所提供的總耗能值，都是藉由週期性地讀取系統平台上 ADC 提供的電壓值、電流值計算而成，使用 ADC 來評估耗能和透過建立功率模型 (power model) 或能量模型 (energy model) 來評估耗能的方式比起來，使用 ADC 的優點是 (1) 較為簡單，因為不必知道硬體事件 (如: pipeline stall) 的耗電量，(2) 較接近硬體的實際耗電，因為是對硬體做直接量測；但其缺點是 (1) 因為需藉由軟體週期性的累計耗電量，因此容易造成系統上額外的效能和耗能負擔，(2) 也因為是由軟體累計耗電量，因此取樣頻率無法超過處理器的運作頻率，使得系統中的每個事件無法被精準掌握，(3) 量測的目標會受到硬體限制，如在 ARM11 MPCore 開發平台上的 ADC 僅提供量測一整個處理器 (包含 4 顆處理器核心) 的耗電量，無法分開量測各個處理器核心，使得 mProfiler 無法進一步將系統的耗能分別歸類給不同的執行緒或是不同的處理器核心。由於使用 ADC 評估耗電有上述缺點，因此本計畫的第二年考慮透過建立功率模型 (power model) 或能量模型 (energy model) 的方式來評估目標程式的耗電量。

參考文獻

[1] 朱守禮, "設計適合高效能 MPSoC 架構的新型低耗能排程機制," 國科會專題研究計畫, 2006.

[2] 伍朝欽, "改善單核心超純量處理器執行效能之研究," 國科會專題研究計畫, 2006.

[3] 伍朝欽, "單晶片多處理機中改善記憶體存取效能之研究," 國科會專題研究計畫, 2005.

[4] 單智君, "ARM 嵌入式系統低耗電匯流排設計," 國科會專題研究計畫, 2001.

[5] 楊佳玲, "省電與性能最佳化技術:從應用面至系統面之探討─子計畫三：考量能量之網路架構晶片軟硬體共同合成架構," 國科會專題研究計畫, 2006.

[6] 黃泰一, "省電與性能最佳化技術:從應用面至系統面之探討─子計畫六：即時系統之動態電源調整與輸出入裝置的整合機制," 國科會專題研究計畫, 2006.

[7] 呂俊欣, "Power-Aware 作業系統技術之研究," 長庚大學資訊工程研究所, 2004.

[8] Berkeley RAMP project, http://ramp.eecs.berkeley.edu/

[9] John L. Hennessy and David A. Patterson, "Computer Architecture – A Quantitative Approach," 3rd edition, Morgan Kaufmann Publishers, 2003, pp. 42-44.

[10] J. Flinn and M. Satyanarayanan, “Powerscope: A Tool for Profiling the Energy Usage of Mobile Applications,” Proc. IEEE Workshop Mobile Computing Systems and Applications (WMCSA 1999), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 2-10.

[11] D. Shin et al., “Energy-Monitoring Tool for Low-Power Embedded Programs,” IEEE Design and Test of Computers, 19(4), 2002.

[12] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,” Proceedings of the 27th International Symposium on Computer Architecture (ISCA), June 2000.

[13] W. Ye, N. Vijaykrishan, M. Kandemir, and M. J. Irwin, “The Design and Use of Simple-Power: A Cycle-Accurate Energy Estimation Tool,” Proceedings of the Design Automation Conference, June 2000.

[14] The SimpleScalar-Arm Power Modeling Project. http://www.eecs.umich.edu/~tnm/power/

[15] V. Tiwari, S. Malik, and A. Wolfe, “Power analysis of embedded software: A first step toward software power minimization,” IEEE Trans.VLSI Syst., vol. 2, pp. 437–445, Dec. 1994.

[16] V. Tiwari, S. Malik, A. Wolfe, and M.T.C. Lee, “Instruction level power analysis and optimization of software,” J. VLSI Signal Processing, vol. 13, no. 2, pp. 1-18, 1996.

[17] A. Sinha and A.Chandrakasan, “JouleTrack – A Web Based Tool for Software Energy Profiling,” Proc. 38th Design Automation Conference, June 2001.

[18] T. K. Tan, A. Raghunathan, G. Lakshminarayana, and N. K. Jha, “Highlevel Software Energy Macro-modelling,” in Proc. ACM/IEEE Design Automation Conference, Las Vegas, Nevada, USA, June 2001.

[19] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak, “Function-level power estimation methodology for microprocessors,” in Proc. Design Automation Conf., June 2000, pp.

810–813

[20] T. Tan, A. Raghunathan, and N. Jha, "Embedded Operating System Energy Analysis and Macro-Modeling," International Conference on Computer Design, pp. 515-222, 2002.

[21] A. Acquaviva, L. Benini, and A. Ricco', "Energy Characterization of Embedded Real-Time Operating Systems," in L. Benini, M. Kandemir, J. Ramanujam, Compilers and Operating Systems for Low Power, Kluwer Academic Publishers 2003.

[22] R. Dick, G. Lakshminarayana, A. Raghunathan, and N. Jha, "Analysis of Power Dissipation in Embedded Systems using Real-Time Operating Systems," IEEE Transactions on CAD, Vol. 22, no. 5, pp. 615-627, May 2003.

[23] RealView™ Emulation Baseboard HBI-0140 Rev D User Guide , ARM, October 2007.

[24] Core Tile for ARM11 MPCore HBI-0146 User Guide, ARM, September 2006.

[25] ARM11 MPCore Processor Revision r1p0 Technical Reference Manual, ARM, February 2008.

[26] Daniel P. Bovet, Marco Cesati, “Understanding the Linux Kernel,” 3rd edition, O'Reilly, November 2005.

[27] M.-L. Li, R. Sasanka, S.V. Adve, Y.-K. Chen, and E. Debes, "The ALPBench Benchmark Suite for Complex Multimedia Applications," in the Proceedings of the IEEE International Symposium on Workload Characterization (IISWC-2005), October 2005.

[28] Chun-Hao Hsu, Jian-Jhen Chen, and Shiao-Li Tsao, "Evaluation and Modeling of Power Consumption of a Heterogeneous Dual-Core Processor," in the 13th International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, Taiwan, Dec. 2007. (EI)

計畫成果自評

本計畫於第一年執行期間提出一套多核心嵌入式系統之效能分析與耗能評估方式，且將其實作成一套適用於多核心嵌入式系統的效能與耗能之監測與分析工具「mProfiler」，藉此幫助多核心嵌入式系統開發者有效掌握多執行緒程式的效能與耗能，該工具除了提供軟體執行緒層級平行度、負載平衡與軟體效能瓶頸等三項效能分析之外，也提供處理器的耗能評估，讓多核心嵌入式系統開發者可以依據此工具的分析結果，對多核心嵌入式系統的軟硬體設計進行最佳化的改良。除此之外，mProfiler 也可以再被進一步的擴充，使其具有自動找尋最佳化能源效率 (energy efficiency) 之多核心處理器架構的功能，如自動找出目標嵌入式程式在何種多核心處理器平台上運作能讓程式的能源效率最佳化，透過本計畫第一年開發的 mProfiler 可以有效找出任一多執行緒嵌入式工作的執行緒層級平行度、執行緒間的溝通機制負擔、每個執行緒的工作量、負載平衡狀況、以及處理器消耗的功率等資訊，有了上述的參考資訊便可以推算出最適合該嵌入式軟體的多核心處理器平台，進而達到高能源效率的目的。

本計畫第一年期間，我們已提出一個適用於異質雙核心處理器 TI OMAP5912 的耗能模型 (power model) ，並發表至國際學術研討會[28] ，而本計畫於 ARM11 MPCore 平台上開發之適用多核心嵌入式系統的效能與耗能之監測與分析工具「mProfiler」，也將在近期內整理成為論文並提出發表。

本計畫第二年預期將會利用 FPGA 處理器平台取代第一年使用之多核心處理器晶片平台，以提供多核心嵌入式系統開發者更具彈性的多核心處理器硬體架構，同時也會繼續改良第一年提出之 mProfiler 的效能分析與耗能評估方式，讓多核心嵌入式系統開發者能更快速地開發出高能源效率的軟硬體設計。