即時嵌入式系統之低功率排程

劉智文、黃駿賢、高新傑、熊博安
國立中正大學資訊工程學系 嵌入式系統實驗室

一、簡介

在即時嵌入式系統中，必須在「時間限制條件」與「電池時效」間有所取捨－以完成相同的工作量而言，在較短時間內可完成工作的即時系統需使用較多的電量；而低功率排程則需要較長的時間完成工作，亦即以較長的執行時間換取低能量消耗。這篇文章主要就是探討在這個取捨下，如何在滿足「時間限制條件」下使「耗電量」降到最低。

以下是一個說明用的例子，假設即時嵌入式系統在無線感應器網路節點(wireless sensor network node)有兩個週期性的工作 r_1 與 r_2，r_1 包含三個子工作 t_1、t_2 與 t_3，分別為傳送資料、接收資料以及傳送確認訊息；r_2 也包含三個子工作，分別為 t_4、t_5 與 t_6，負責在三條軸線上移動感應器位置。t_1、t_2 與 t_3 皆須使用相同的網路裝置 k_1，而 t_4、t_5 與 t_6 皆須使用相同的馬達裝置 k_2。因此，在這個例子中，我們必須將這幾個子工作排程，使得它們在滿足各自的時間條件限制下，消耗的功率能降到最低。在這篇文章中，我們提出一個滿足這種行為的模型和演算法－即各個工作以最低的功率消耗，在各自的時間限制內完成工作。

傳統上一般都使用 Earliest Deadline First(EDF)和 Rate Monotonic Scheduling[1] (RM)兩個演算法對即時性的工作做排程，但這兩個演算法有某些限制條件存在，例如當一個系統模型由非互相獨立的週期性工作集合所構成時，這兩個演算法所採用的工作圖模型便無法完全地表達此嵌入式系統的行為，因此較複雜的模型是必要的。在此，我們使用一個以延伸派翠網路(Extended Petri Nets)為基礎的模型。在過去的一個文獻[2]中，Real Time Petri Nets (RTPN)被用來描述一個工作集合，此集合含有一組週期性的工作，每個工作又包含有先後順序的子工作，子工作有區域性的時間限制條件，而整個 RTPN 有全域性的時間限制條件。Quasi-dynamic scheduling (QDS)[2]可對一個有區域性時間限制條件、全域週期以及全域性時間限制條件的 RTPN 集合做排程。在 QDS 中，我們將這些在執行期間依資料做分枝判定(branch)的工作分解成不同的行為設定(behavior configurations)，並且對它們做半靜態的排程。再對每一個被分解的行為設定做動態排程，以滿足每個子工作的區域性截止期限(local deadline)，及它們之間的先後順序和全域性截止期限(global deadline)。

這篇文章的其餘部分組織如下：在第二節中，先介紹一些過去應用在即時嵌入式軟體合成上的低功率消耗排程方法；我們的系統模型架構在第三節中展示，並以一個實際的例子作為說明；在第四節中描述我們提出的排程演算法；第五節，以兩個例子證明我們所提出的演算法在效能上有較優異的表現；最後，第六節為這篇文章做個簡單的結語。

二、相關的技術文獻

以編譯器為基礎的技術[3、4]已經被提出用來降低嵌入式軟體的功率消耗；可變頻變壓的處理器則透過工作排程演算法(task scheduling algorithm)[5、6]來降低嵌入式系統之的功率消耗；再者，裝置排程(device scheduling)也是一種主要降低功率消耗的技術。作業系統所執行的功率管理－即一般大家熟知的動態功率管理

(dynamic power management)，指的是一組工作在其執行時期(run-time)被排程，用來滿足它們的執行時間限制與減少功率消耗。輸入/輸出裝置(I/O device)的動態功率管理常常是使用預測(predictive)與隨機(stochastic)模型[7、8]，但對於嚴格即時嵌入式系統(hard real-time embedded system)而言，這兩種模型是不夠精確的。雖然線上裝置排程(on-line device scheduling)[9、10]是一個很普及的作法，但某些缺點依然存在，其中一個缺點便是排程器需要時間去決定一個龐大工作集合的排程。而離線裝置排程(off-line device scheduling)則是產生一個事先排好的工作執行順序，使得這些工作執行時，完全滿足它們的時間限制；然而離線裝置排程方法並不如線上裝置排程那般被廣為討論。Swaminatham[11] 提出一個針對工作圖模型(task graph model)的離線裝置排程技術，reachability tree 則被用來找出所有排程並輸出消耗功率最低的排程。

在這篇文章中，我們提出半動態裝置排程(quasi-dynamic device scheduling)技術，有別於過去被使用最多的 EDF 排程演算法，我們使用半動態排程(quasi-dynamic scheduling)方法。在後面的章節會將我們提出的方法與過去的方法比較，包含時間、記憶體和消耗功率的比較。

三、系統模型與目標問題

甲、系統模型

我們提出 Power-Aware Real Time Petri Net(PARTPN)模型來描述低功率即時嵌入式軟體的行為。

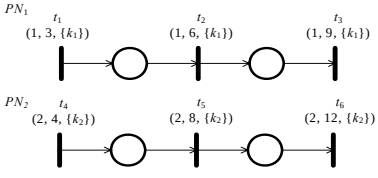
定義一、Power-Aware Real Time Petri Net (PARTPN)由七個元素所組成如下，

$(P, T, F, M_0, \tau, p, d)$：

1. P：一組有限的 places。
2. T：一組有限的 transitions，
   $P \cup T \neq \phi$，$P \cap T = \phi$。
3. F： $(P \times T) \cup (T \times P) \rightarrow N$ 是一個在 place 與 transition 之間的 weighted flow relation。
4. M_0：$P \rightarrow N$ 指的是初始狀態，也就是 place 上有幾個標記(token)。
5. τ ： $T \rightarrow N \times (N \cup \infty) \times K$, i.e., $\tau(t) = (\alpha, \beta, L)$，其中 $t \in T$，α 是 transition 的最長執行時間(Worst Case Execution Time)，β是 transition t 的區域截止期限(local deadline)，$L \subseteq K$ 是在執行 transition t 時所需要的一組裝置清單，K 代表可被排程的一組系統輸入輸出裝置。我們分別使用縮寫 $\tau_\alpha(t)$, $\tau_\beta(t)$,及 $\tau_L(t)$來表示 transition 的執行時間，區域截止期限，及一組需要的裝置。
6. $p \in N$ 指的是全域週期(global period)，而 $d \in N$ 是整個 PARTPN 的全域截止期限(global deadline)。

乙、實例說明

接下來以無線感測器節點（wireless sensor node）上的軟體舉例說明。這些感測器節點必須移動位置並傳送資料給本身以外的節點（見圖一），圖一中圓圈表示 place、垂直線表示 transition、垂直線上方的數字表示 transition name 和相關的屬性(attribute)、箭號表示 arc、箭號上的數字表示於 F 定義的權重(weight)。其中，transition 的屬性包含三個參數：執行時間（execution time）、截止期限（deadline）、transition 所需的裝置集合（a set of devices），以圖中 transition t_1 來說，(1, 3, $\{k_1\}$)分別表示 t_1 的執行時間為 1 個時間單位（time unit）、時間限制為 3 個時間單位、及 t_1 需要裝置 k_1。表一列出所有 I/O 裝置的屬性。



圖一、感測器網路節點之 PARTPN 模型

表一、感測器網路節點之裝置屬性

	δ_w	δ_s	δ_{wup}	δ_{sdp}	δ_{wut}	δ_{sdt}
k_1	5	1	3	3	1	1
k_2	5	1	3	3	1	1

丙、目標問題(target problem)

我們想解決的問題是，如何達到由 PARTPN 系統模型描述的需求下，可以滿足所有使用者所給的限制條件，也就是全域及區域截止期限(global and local deadline)，系統的總記憶體用量，及總能量使用的上限。

四、低功率半動態排程演算法

為解決上述即時嵌入式軟體合成的問題，我們提出低功率半動態排程(Low-Power Quasi-Dynamic Scheduling)演算法，以下簡稱 LQS。如同 QDS 與其他靜態排程技術(static scheduling techniques)[11]，LQS 也使用 reachability tree，從一個給定的 PARTPN 模型集合中以深度優先搜尋(depth-first search)的方式建構 reachability tree，而 LQS 的前端處理程序即與 QDS 類似。一個滿足所有非時間限制條件的排程我們稱之為 Extended Quasi-Static Schedules (EQSS)[12]。QDS 會產生一個排程耗時最短的 EQSS 排程，而 LQS 則是產生總消耗功率最少的 EQSS 排程，基本的功率計算方式和為了有效率地找到可行的 LQS 排程的 reachability tree 裁剪規則(pruning rules)，將在本節中詳細介紹。

對每個 reachability tree 的節點，系統的資源利用都必須被計算，這裡所指的系統資源包含總時間、記憶體用量和功率消耗。在 reachability tree 的根節點(root node)，亦即初始狀態(initial marking)，總時間、記憶體用量及功率消耗均為 0。時間和記憶體的計算方式是很直觀的，而功率消耗的計算方式如下：給定一個節點 M 和其後繼節點(successor node) M'，在經過由 M 至 M'的 transition t 後，M'上的功率消耗計算方式如下：

1. 一開始，所有的裝置都被設定在工作狀態。我們以 M_0 表示初始狀態，以 $D_w(M_0)$ 表示一組在初始狀態時，處在工作狀態的裝置的集合，並設 $D_w(M_0)$為 K。
2. 在狀態為 M 的一個 transition t 上，最長的裝置甦醒時間(wake up time)$\varepsilon(t, M)$的計算方式如下：

   $\varepsilon(t, M) = \max(\delta_{wut}(k_i)), k_i \in \tau_L(t) \setminus D_w(M)$。
3. 區域的功率消耗 $v(t, M)$是在狀態 M 時，執行 transition t 所消耗的功率的總和。其功率消耗計算方式如下：

$$\sum_{k_i \in D_R(t,M) \setminus D_w(M)} \delta_{wup}(k_i) + \delta_w(k_i) \times P +$$
$$\sum_{k_i \in D_w(M) \setminus D_R(t,M)} \delta_{sdp}(k_i) + \delta_s(k_i) \times Q +$$
$$\sum_{k_i \in D_R(t,M) \cap D_w(M)} \delta_w(k_i) \times (\tau_\alpha(t) + \varepsilon(t,M)) +$$
$$\sum_{k_i \in K \setminus (D_R(t,M) \cup D_w(M))} \delta_s(k_i) \times (\tau_\alpha(t) + \varepsilon(t,M))$$

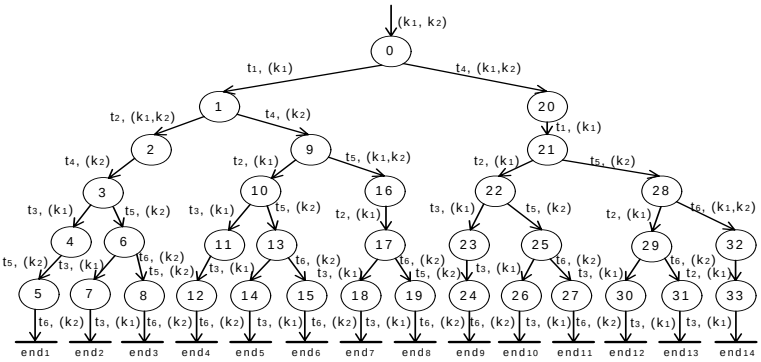
   其中 P 為 $\max\{0, \tau_\alpha(t) + \varepsilon(t, M) - \delta_{wut}(k_i)\}$，$Q$ 為 $\max\{0, \tau_\alpha(t) + \varepsilon(t, M) - \delta_{sdt}(k_i)\}$，$D_R(t, M)$則是在狀態 M 時，執行 transition t 所需要的一組處在工作狀態的裝置，這組裝置有可能是一直都處在工作的狀態或是剛甦醒準備工作，之後會對此詳加介紹。

以下介紹 LQS 中的一些規則，用來決定何時一個 transition 應該被執行以及哪些裝置需要被喚醒。

1. 在狀態 M 時，如果想在執行 transition t_1 後產生一個後繼節點(successor node)，必須先檢查在 G 中所有其它被致能(enabled)的 transition t_2 的可排程性(schedulability)，其中 G 是一組同時被致能的 transitions 的集合。在 G 這個集合中，我們說 transition t_1 相對於其他 transition t_2 可以被選上而開始執行的條件是：

   $\rho_\beta(t_2) - (\rho_\alpha(t_1) + \varepsilon(t_1, M)) \geq \rho_\alpha(t_2)$，

   其中$\rho_\beta(t_2)$指的是離 t_2 的區域截止期限還剩多長的時間，$\rho_\alpha(t_1)$則是 t_1 還剩多少時間可以執行。



圖二、感測器網路節點之 Reachability Tree

2. 在狀態 M 時，假設 transition t_1 被選上可以執行，此時可能有一些正處在工作狀態的裝置是執行 t_1 時不會用到的，以下我們特別指出所有滿足下述不等式的這些裝置必須保持在工作狀態，$\rho_\alpha(t_2) \le \rho_\beta(t_2) - (\rho_\alpha(t_1) + \varepsilon(t_1, M)) < \rho_\alpha(t_2) + \varepsilon(t_2, M')$，

   其中 M' 是從狀態 M 執行了 transition t 後所到達的狀態。因此，在狀態 M 執行了 transition t 後，仍必須保持在工作狀態的裝置如下所示：

   $D_R(t, M) = \tau_L(t) \cup \cup_{t'} \tau_L(t')$ 。

五、真實應用範例

首先，我們以第一節中介紹的感應器網路節點(sensor network node)內的即時嵌入式軟體為例子解說 LQS，第三節中已為此範例建好兩個 concurrent PARTPNs。使用 LQS 應用程式所產生的 reachability tree 如圖二所示，關於 reachability tree 的詳細建構方式我們在此省略不談。圖二中，reachability tree 建構出 14 個排程，其中排程<t_1 t_2 t_4 t_5 t_6 t_3>有最低的功率消耗－僅 66 單位，及花費 10 個時間單位，因此 LQS 應用程式將依此排程產生程式碼。

使用裁剪技術(pruning technique)後，此例只產生包含 34 個節點的 reachability tree，假若沒有使用裁剪技術，則此例將產生一個包含 69 個節點的 complete reachability tree，推想可知，若沒有使用裁剪技術，則規模較大的系統的 reachability tree 將會擁有數量更為龐大的節點總數。

為了證明我們所提出的 LQS 演算法較具優勢，我們將 LQS 與另一種也是以建構 reachability tree 尋找低功率排程的演算法－Energy-Optimal Device Scheduler（EDS）[11] 做比較。表二列出 LQS 與 EDS 各方面的數據結果，我們不難發現使用 LQS 做出來的排程結果
不但在 reachability tree 的節點數量上優異許多，其低功率消耗排程結果也優於 EDS。

表二、感測器網路節點之 LQS 與. EDS 比較

方法	排程	時間	記憶	功耗	節點數
LQS	<$t_1t_2t_4t_5t_6t_3$>	10	2	66	34
EDS	<$t_1t_4t_2t_3t_5t_6$>	14	N/A	78	371

我們將 LQS 應用到另一個例子－無線藍芽(Bluetooth)通訊協定裡的主從角色交換(Master/Slave role switching)。兩個藍芽裝置 A 與 B 交換角色的程序可以用 4 個 PARTPNs 將模型建好，但礙於篇幅，省略此 4 個 PARTPNs 模型的介紹。A 與 B 在做角色交換時，必須於 host 層與 host-control/link manager 層兩層間通訊，這兩層分別有 4 個和 9 個 untimed EQSS 排程，因此總共有 36 種系統行為，即共有 36 種 reachability tree 將被建構，但只有一種系統行為可被 LQS 排程。LQS 於此例中產生的

reachability tree 包含 34 個節點和 6 個可行的排程(feasible schedule),其中消耗功率最低的排程需要 189 個功率單位及 24 個時間單位,假若沒有使用我們提出的裁剪技術,則產生的 reachability tree 將包含 8191 個節點。

六、結論

我們針對即時嵌入式軟體提出一個以裝置導向(device-centric)的低功率消耗排程演算法。首先,我們提出 Power-Aware Real-Time Petri Net (PARTPN)模型來描述嵌入式系統的行為,Low-Power Quasi-Dynamic Scheduling(LQS)則是我們用來找出此模型中消耗功率最低排程的主要演算法,它不只可以保證所產生出來的排程滿足所有子工作的區域性時間限制條件、所有工作的全域性時間限制條件,以及這些工作在記憶體和功率消耗上的限制,其消耗的功率也是最低的。與離線輸入/輸出裝置排程(off-line I/O device scheduling)相比較,我們提出的方法無論在效率亦或是能力上,都是優異許多的。

參考文獻

1. C.-L. Liu and J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the ACM*, **20**(1) (1973) 46–61.
2. P.-A. Hsiung and C.-Y. Lin, Synthesis of real-time embedded software with local and global deadlines, in *Proc. IEEE/ACM International Symposium on Hardware-Software Codesign and System Synthesis*, (ACM Press, USA, 2003), pp. 114–119.
3. O. Acevedo, A survey of software optimization techniques for low-power consumption, in *Proc. Computer Research Conference*, (2002).
4. Y. Cao and H. Yasuura, A system-level energy minimization approach using datapath width optimization, in *Proc. 2001 International Symposium on Low Power Electronics and Design*, (2001), pp. 231–236.
5. J.-L. Kuo and T.-F. Chen, Dynamic voltage leveling scheduling for real-time embedded systems on low-power variable speed processors, in *Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, (2002), pp. 147–155.
6. N. K. Jha, Low power sytem scheduling and synthesis, in *Proc. of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, (2001), pp. 259–263.
7. P. Kumar and M. Srivastava, Predictive strategies for low-power RTOS scheduling, in *Proc. IEEE International Conference on Computer Design: VLSI in Computers and Processors*, (2000), pp. 343–348.
8. Q. Qiu, Q. Wu, and M. Pedram, Dynamic power management of complex systems using generalized stochastic Petri nets, in *Proc. 37th Design Automation Conference*, (2000), pp. 352–356.
9. V. Swaminathan, K. Chakrabarty and S. S. Iyengar, Dynamic I/O power management for hard real-time systems, in *Proc. of the 9th International Symposium on Hardware/Software Co-Design*, (2001), pp. 237–242.
10. Y,-H. Lu, L. Benini, and G. D. Micheli, Low-power task scheduling for multiple devices, in *Proc. 8th International Workshop on Hardware/Software Co-design*, (2000), pp. 39–45.
11. V. Swaminathan and K. Chakrabarty, Pruning-based energy-optimal device scheduling for hard real-time systems, in *Proc. of the 10th International Symposium on Hardware/Software Co-Design*, (2002), pp. 175–180.
12. F.-S. Su and P.-A. Hsiung, Extended quasi-static scheduling for formal synthesis and code generation of embedded software, in *Proc. 10th International Symposium on Hardware/Software Co-Design*, (2002), pp. 211–216.