Profile-Based Technique for Dynamic Power Management in Embedded Systems

C. Magesh Kumar M. Sindhwani T. Srikanthan
Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore

Abstract

One of the key challenges of embedded system design is the management and conservation of power. Static power reduction methods such as synthesis of efficient hardware and compilation for low power are applied at design time. In contrast, Dynamic Power Management (DPM) techniques leverage on the runtime characteristics to reduce power when systems are serving light workloads or are idle. Though much work has been done on profile-based CPU power management, similar efforts are relatively less for I/O devices. This paper proposes an algorithm for profile based power management of the peripherals for embedded computing systems. It enables suitable decisions for efficient operation of the peripherals considering both power and performance. The algorithm lends well for applications with both single and multiple independent peripherals.

1. INTRODUCTION

Achieving highly energy efficient computation has become one of the major challenges in designing future electronics systems. The increase in user demands for mobile and embedded systems technology requires an equivalent increase in performance that causes an increase in the power consumption of these devices. Low power consumption is required to achieve acceptable autonomy in battery-powered systems and operation cost of stationary systems as well as to reduce the environmental impact (e.g. heat dissipation, cooling-induced noise). Static power reduction techniques are applied at design time. These include power efficient hardware synthesis and software compilation and synthesis. Another important energy conservation approach and also an attractive alternative to inflexible hardware solutions is Dynamic Power Management (DPM) [1].

The fundamental premise for the applicability of DPM is that systems (and their components) experience non-uniform workloads during operation time [2]. Thus, DPM encompasses a set of techniques, applied at run-time, that achieve energy-efficient computation by selectively turning off (or reducing the performance of) system components when idle (or partially unexploited).

Modern hardware designs provide several features to support DPM [2] [3]. These features include multiple power states in I/O devices and variable-voltage processors. Given these features, a DPM scheme can make decisions on changing the operating voltage of a processor, and switching devices to low-power (sleep) states during periods of inactivity. The transition between various modes of operation has overheads that span the tradeoff between performance and power. The critical issue in dynamic power management is to ensure that by scaling down the modes of operation, we still have justifiable overall energy savings.

Application aware strategies for energy centric partitioning and management will lead to efficient DPM solutions for dedicated systems. Several efforts have been reported in literature for CPU-centric techniques for profile-driven power management [4] [5], but the feasibility of such techniques has not been often addressed for power management of I/O subsystem.

This paper addresses one such profile based technique to realize power management of peripheral devices (I/O) in embedded systems. The proposed algorithm analyzes the control flow information of an application having I/O accesses of certain peripherals. The Control Flow Graph (CFG) is used in many different analyses and optimization throughout modern compilers [6] [7] and is of central concern for the proposed algorithm. In this work, the CFG data structure is analyzed to find the potential idle time durations that are *profitable* for energy savings without violating the performance constraints. This helps in finding suitable locations in the CFG to turn ON/OFF the peripherals.

In Section 2 of this paper, we discuss the importance and use of a profile driven approach and control flow analysis with an emphasis on power management in embedded systems. Section 3 explains the proposed methodology for profile-based peripheral power management with a brief description about the application model used in this work and related power management concept. Efficiency analysis of the overall approach is presented in Section 4 before concluding in Section 5.

978-1-4244-2315-6/08/$25.00 ©2008 IEEE.

2. PROFILE-BASED POWER MANAGEMENT AND CFG

Identifying key characteristics from an application is essential from an embedded system design perspective. It is one of the most used techniques in code/compiler optimization, hardware/software co-design, etc. which aim for more power/area/time-efficiency[6][7].Program development, maintenance and resource usage can be greatly improved by the static and dynamic analyses that provide information about application behavior [8]. The CFG is a fundamental data structure needed by almost all such analysis techniques like global data-flow analysis, the construction of static single assignment graph and data-dependence analysis [9].

The CFG which is a representation through basic blocks and their connecting edges aids in identifying the flow between the blocks. Hence, it provides an idea about the possible flows of execution for a program [10]. There are various commercial and open-source tools [6][7][11] to generate CFGs of the source code by performing static analysis with certain safe assumptions. Having the knowledge of the possible inputs and most critical and/or frequent paths in the design, these assumptions can be refined further.

One such important example which relies heavily on control flow information is the Worst Case Execution Time (WCET) analysis used in hard-real time system design [12]. To derive the execution time of a program, both the characteristics of the program code and the computer hardware must be considered. These techniques are based on certain assumptions like: One Specific program running in isolation, running on a certain CPU & Clock, compiled with a certain compiler, no interfering background activity and no task switches.

Various approaches and techniques reported in literature for software-controlled power management dealing with CPU power consumption make use of the application's control-flow profile [4][13][14]. Energy consumption of CMOS circuits has a quadratic dependence on supply voltage and linear dependence on frequency. Hence, Dynamic Voltage Scaling (DVS) & Dynamic Frequency Scaling (DFS) that changes supply voltage/frequency at runtime is an important technique of power management for the CPU.

Inter-task dynamic voltage scheduling (Inter-DVS) determines the supply voltage on task-by-task basis. With multiple tasks, the Inter-DVS techniques assign the proper speed to each task dynamically while guaranteeing all their deadlines. Intra-task dynamic voltage scheduling adjusts the supply voltage within an individual task boundary. After executing a segment, it adjusts the clock speed and supply voltage exploiting the slack times from the executed segments of a program. In path-based Intra-DVS, the slack times are found using the control flow information [4]. Selecting all the program locations where the changes of remaining workload can be identified, it inserts voltage scaling code at compile time. This code is executed at run-time exploiting all slack times coming from run-time variations of different execution paths.

Techniques such as using the remaining worst case execution path, remaining average case execution path, optimal-case execution path have been proposed in the literature [4] [13]. Another approach by Nevine et al. [14] involves collaborative efforts of Operating System and compiler for voltage scaling. All these techniques rely on the control flow information as the application model and use either CFGs generated randomly or obtained from certain tools like Trimaran/SUIF.

Regarding I/O-centric power management techniques, the rules to determine whether to shut down a device are called policies which are categorized as Time-out, Predictive and Stochastic. Several related works based on power management policies have been reported [2][15]. Energy-conscious I/O device scheduling is another approach targeting real time systems [16][17]. Other efforts for I/O subsystem power management reported in the past are found in [18][19][20]. Our approach attempts to utilize the Control flow profile of an embedded code to take power management decisions of the peripherals rather than the CPU, with due consideration given to both performance and power.

In the next section, the application model used in our work and the proposed methodology for peripheral power management are explained.

3. PROPOSED METHODOLOGY

In this section, we first present two main concepts as applicable to our work- Application Model and Break-Even Time. Thereafter, we provide a detailed explanation of the proposed methodology for profile-based power management of the peripherals.

A. Application Model:

In this work, the CFG data structure with possible edges & nodes represents the embedded application and the choice of abstraction level at which it is obtained may differ. We consider the general form of an embedded application that may consist of sequential code, branches, loops and procedures. Here, only natural loops are considered as most modern compilers of high level languages generate only natural loops, having exactly one entry point i.e. the loop header which dominates the loop body [10].

Essentially, this CFG contains nodes of CPU related operations and peripheral (I/O) related operations.

Here the number of I/O nodes and their positions are chosen arbitrarily leaving the rest as CPU-related nodes. The nodes of the control flow graph are given weights which are directly related to the time taken for execution of each basic block. For simplicity, we assume that each basic block has fixed time duration for its execution and all edges have equal probability of transition. In practice, we might get various execution times for the basic blocks which stem from a number of different reasons such as data dependencies, hardware effects such as pipelining and caching, etc. In such cases, the execution time estimate of a basic block for a particular target processor and the more frequent edges can be obtained by profiling runs and can be given as additional inputs to the algorithm.

B. Concept of Break-Even Time:

An important device feature used in dynamic power management is Break-even time. Power management is a prediction problem; it seeks to forecast whether an idle period will be long enough to compensate for the overhead of power state changes. The minimum length of an idle period to save power is called the *Break-even time* (T_{BE}) [2].

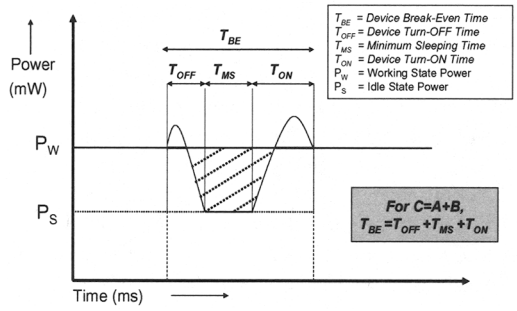

Figure 1. Break-Even Time Explanation

Let a device have a state transition delay of *To* (shutdown and wake-up delays), transition energy of *Eo* and its power consumption in the working and sleeping states be *P*w and *P*s. In Figure 1, the areas A and B represent the energy overshoots consumed during shut-down and wake-up of the device. To compensate for the energy consumed in changing states, the device must stay in the sleeping state at least for duration of T_{MS} termed as the minimum sleeping time. In short, the break-even time makes energy consumption in both cases equal.

$$Pw \times T_{BE} = Eo + Ps \times (T_{BE} - To) \quad (1)$$

The break-even time has to be larger than the transition delay. So, T_{BE}= max [(*Eo* − *P*s × *To*)/(*P*w − *P*s), *To*]. The sum of device turn ON time $\mathbf{T_{ON}}$, required minimum sleeping time $\mathbf{T_{MS}}$ (when applicable) and device turn OFF time $\mathbf{T_{OFF}}$ is termed as the Break-even time $\mathbf{T_{BE}}$.

Hence,

$$T_{BE} = T_{OFF} + T_{MS} + T_{ON} \quad (2)$$

C. Analyzing CFG for DPM

The application is modeled as described earlier with a CFG which has I/O-access blocks at various arbitrary locations. To analyze the CFG, an offline algorithm has been devised to decide about the dynamic power down of peripherals during their idle times for power management. The algorithm consists of two steps: Identifying appropriate Switch-ON points ensuring performance and identifying profitable Switch-OFF points to get energy savings.

Here, Switch-ON points are denoted as "A" points with "A" symbolizing "Active" state of operation and Switch-OFF points are denoted as "L" points - with "L" symbolizing "Low" power idle state. And whenever the need arises to refer both of them, the term "Switching point" is used. By referring to "place" a switching point, what we mean here is to "identify" the CPU-related basic block in the CFG which favors the location of a control word for switching the peripheral to ON or OFF. Locating the switching point further within the basic block depends on the granularity of the block sizes considered. For negligible block sizes compared to the device features, we can safely locate the switching point at the start of a block.

To start with, the peripheral under consideration is assumed to be in the OFF state. In the first pass of the algorithm, the CFG data structure is analyzed to get all necessary points for turning ON the peripheral. To turn it ON, a Switch-ON point ("A") is "placed" along every path which leads to that I/O block at a location in the CFG which has "enough time" (i.e. T_{ON}) to Turn-ON the peripheral without performance penalty. This ensures that the device will be turned ON along all possible entries of an I/O access.

In the second pass, now having known the locations of peripheral accesses and their Switch-ON points in the CFG, suitable points to turn OFF the peripheral for energy savings are identified. Among all possible paths from one I/O access to the successive I/O access, if the shortest path has "profitable time" (>T_{BE}) to Turn-OFF the peripheral without power penalty, then a Switch-OFF point ("L") is "placed" along every path from the current I/O block to the next I/O block.

Figure 2 summarizes the steps of the algorithm. The algorithm has two functions performed in two passes. The function "place_A" locates suitable positions in the CFG for timely Switch-ON and the function "place_L" identifies the positions for profitable Switch-OFF of the device.

Algorithm

```
Input: CFG, T_ON, T_BE
P(n): Immediate Predecessors of block 'n'
S(n): Immediate Successors of block 'n'
D(n): Dominators of block 'n'
IO  : Set of all I/O blocks
SD(m,n):Shortest Distance from block 'm'to'n'
First Pass:
Procedure: place_A()
for all  io ∈ IO
{
temp x=io
iteration_1()
 { push all p∈P(x)
   while (stack ≠ null)
   { p=pop
     if p∉IO and x∉D(p)
       if SD(p,io)≥T_ON
        place "A" in block 'p'
       else {
        x=p
        call iteration_1() }
   }
 }
}
Second Pass:
Procedure: place_L()
for all  io ∈ IO
{
temp y=io
iteration_2():
 { push all s ∈ S(y)
   while (stack ≠ null)
  {  s=pop
     if s∉IO and s∉D(y)
       if SD(s,IO)>T_BE  (check ∀ io ∈ IO)
         place "L" in block 's'
        else{
         y=s
         call iteration_2()}
  }
 }
}
for all  a ∈ A (got from first pass)
{
temp z=a
iteration_3()
 {
  push all s ∈ S(z)
  while (stack ≠ null)
  { s=pop
    if s∉IO and s∉D(z)
      if SD(s,IO)>(T_BE+T_ON) (check ∀ io ∈ IO)
         place "L" in block 's'
      else{
         z=s
         call iteration_3()}
  }
 }
}
```

Figure 2. Algorithm to Analyze CFG

1) *Identifying Switching Points:* The two steps of this algorithm are elaborated in detail as follows:

a) Performance-centric Identification of Switch-ON points ("A"): Each peripheral usage will have its own performance constraint as to within what duration of time, a peripheral has to respond for its access. Therefore, the time taken for a device to get turned ON, T_{ON}, determines the points to switch it ON.

The Switch-ON points are placed before every possible entry to all peripheral accesses at a distance greater than or equal to T_{ON}, thereby ensuring that the device gets turned ON along all possible traces of the program execution. Figure 3 serves as an example to explain how the Switch ON points ("A") are found out. Here the sizes of all blocks are taken to be equal. As shown, every path leading to the I/O block has an "A" point at a distance greater than or equal to T_{ON}.

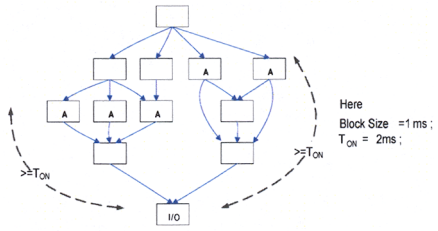

Figure 3. Sample scenario to show the placement of "A" points

b) *Profitability-centric Identification of Switch-OFF points ("L"):* As a device is turned off when it is idle, the energy savings obtained by turning it OFF should be justified. For that, the cost involved in turning it OFF and turning it ON again when it is required has to be compensated. Hence there will be a Switch OFF point, only if, there is a profitable time for energy savings before the next access of I/O takes place. The break-even time T_{BE} determines the minimum time duration between a Switch-OFF point and an I/O access. Figure 4 depicts a sample scenario for "L" points after an I/O service.

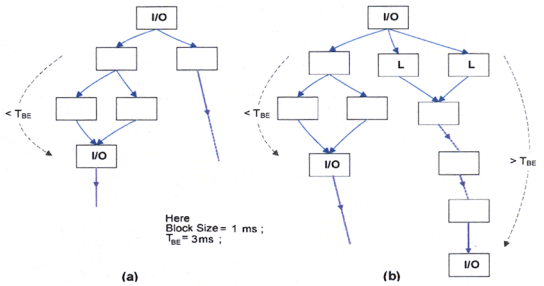

Figure 4. Sample scenario showing the placement of "L" points after I/O (a) Edge on the left not satisfying T_{BE} constraint- Hence no "L" point (b) Edge on the right satisfying T_{BE} constraint - Hence placing "L" Point

Since from step 1, the Switch ON points get placed before every peripheral access, there will be a Switch-ON point between a Switch OFF point and the next possible peripheral access along all paths of the program execution. This ensures that the device will be

turned ON when it is required again without violating performance constraints. Also, there is a possibility that once a device gets switched ON from a "A" point (placed in step-1), the program execution may take some other alternate routes of execution (which doesn't have an I/O within T_{BE}) leaving that "A" point unnecessary. Hence again, there arises a necessity to put an "L" point after identifying such alternate paths of execution. Figure 5 depicts this scenario. This is also one of the reasons for considering the switch-OFF points in the second step of the algorithm rather than in the first.

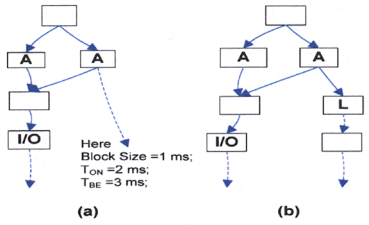

Figure 5. Sample scenario showing the placement of "L" points after "A" points (a) "A" point on the left leads to I/O- Hence no "L" point (b) "A" point on the right may not always lead to I/O -Hence placing "L" point

2) *Loops and procedures:* While calculating the shortest path from any switching point to an I/O access, all loop iteration values are safely given a minimum loop bound of 1. In real scenarios, if the minimum loop bounds for various loops are known, these can be given as additional inputs and accordingly the shortest path algorithm can be modified to account this additional data.

If a procedure or subroutine is selected as the switching point, there arises an issue because the same procedure can be called by two or more paths which may not require the Switch-ON/OFF decision to be taken. This problem can be tackled by inserting virtual blocks (with weights based on the respective subroutine's execution time) in the places of all subroutine calls. Once a virtual block is selected as the switching point, the predecessor or the successor of that virtual block is selected with a certain penalty in either power or performance.

3) *Application accessing Multiple I/O Devices:* The same algorithm can be used for applications accessing multiple peripherals if they are independent of each other. For such case, its CFG can be first identified with all types of I/O accesses and can be categorized. The algorithm then will give as many sets of switching points as the number of peripherals. To ensure that a switching point doesn't get placed in an I/O access block of any other device, the switching point is selected as either the predecessor or successor with a certain penalty in performance or power.

4. EFFICIENCY ANALYSIS

This section discusses the runtime efficiency of the proposed approach with techniques to manage the runtime overheads and to estimate the energy savings determining the applicability of this approach.

A. Managing Overheads for Efficiency:

During program execution, there will be some additional work done in executing the control words for these switching points. Hence there is an associated cost function for these overheads incurred by the CPU/RTOS. These additional overheads can be accounted by varying the device parameters i.e. by expressing the overhead cost in terms of increased break-even time in the algorithm. Also, an optimum set of "Switching points" can be chosen which have relatively more significant energy savings to justify the overheads. This improves the efficiency of the overall approach. This also helps in knowing the feasibility and applicable scenarios of this proposed method. For example, the switching points found offline can be fed back into the compiler, making it similar to static power management techniques. As a more dynamic approach, a monitoring task can be assigned to check whether current program execution is matching/nearing the switching points of interest. The CPU/RTOS overheads and performance penalty of these approaches can be studied beforehand and accordingly the parameters of the algorithm can be adjusted. In addition, by choosing only an optimum set of significant switching points the overheads can be reduced further.

B. Estimation of Potential Idle Time Durations:

After accounting for the overheads through increased break-even time, an estimation of potential idle time durations for energy savings can be calculated. With this proposed method, the worst-case potential idle time duration for energy savings depends upon the shortest time duration taken by a Switch OFF point to meet the nearest Switch ON point. Hence, the energy savings is much dependent upon the density of I/O access blocks. Less closer they are, more opportunities for energy savings. Hence an application with relatively larger distances among consecutive I/O access blocks along several paths of execution can gain much from this approach. The time duration between a Switch OFF point and the nearest successive I/O access block along a path is a measure of the idle time between successive I/O blocks.

Denoting that quantity as T_{SI}, the time duration that proves fruitful for energy savings (T_{ES}), will be the difference between T_{SI} and the break-even time (T_{BE}).

$$T_{ES} = (T_{SI} - T_{BE}) \quad (3)$$

The cumulative sum of T_{ES} along a dynamic trace of execution gives a quantitative measure of the energy savings. Considering percentage energy savings for an average of T_{ES},

$$T_{ES} / T_{SI} = (1 - T_{BE} / T_{SI}) \quad (4)$$

Defining the ratio T_{BE}/ T_{SI} as the "density index" d,

$$T_{ES} / T_{SI} = (1 - d), \quad 1 > d > 0. \quad (5)$$

The quantity "d" gives us a measure of the "density" of I/O access blocks in a CFG. Lesser the value of "d", sparser are the I/O access blocks favoring the applicability of this approach. Figure 6 represents the equation (5) graphically.

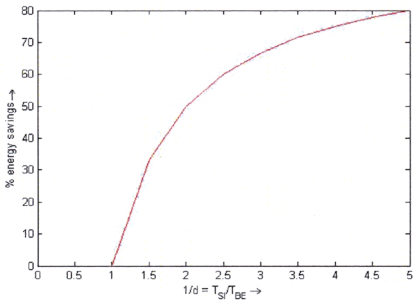

Figure 6. Percentage Energy Savings Vs Sparseness of I/O accesses

Techniques like assigning weights for paths depending upon their frequency of execution will give a clearer picture of the energy savings for a real application. Average energy savings can be calculated by choosing a few key frequent/critical paths accounting the idle time obtained from each path. For applications accessing multiple peripherals, the energy savings achieved for the individual devices contribute to the cumulative energy savings of the system.

5. CONCLUSION

In this paper, an algorithm is proposed which manipulates the control flow information of an embedded application profile for peripheral power management. The algorithm finds appropriate locations in the CFG for placing switching points. These switching points enable the timely switch-ON of an I/O device guaranteeing performance and the profitable switch-OFF of the device when it is idle. The algorithm handles applications accessing both single and multiple independent peripheral devices. To improve the overall efficiency of the approach, suitable techniques are suggested to manage the possible runtime overheads. It is evident that significant energy savings can be obtained from this approach depending upon the density of successive I/O access blocks in the application. Efforts are underway to further extend this work for peripheral devices with multiple-power down states.

REFERENCES

[1] L. Benini, A. Bogliolo. G. D. Micheli G D, "System-level Dynamic Power Management", *Proc. of the IEEE Alessandro Volta Memorial Workshop on Low-Power Design, 1999.*

[2] L. Benini et al., "A Survey of Design Techniques for System-Level Dynamic Power Management", *IEEE Trans. VLSI Sys.*, vol. 8, no. 3, June 2000.

[3] Intel, Microsoft, and Toshiba. Advanced configuration and power interface specification. [Online] http://www.acpi.info/

[4] Dongkun Shin, Jihong Kim, "Optimizing Intratask Voltage Scheduling Using Profile and Data-Flow Information", *IEEE Trans. on CAD of Int. Ckts. and Sys.*, vol. 26, no. 2, 2007.

[5] Takanori Okuma, Tohru Ishihara, "Software Energy Reduction Techniques for Variable-Voltage Processors", *IEEE Design and Test of Computers,* Vol. 18, Issue 2, Mar-Apr. 2001.

[6] Trimaran- Compiler Infrastructure www.trimaran.org/

[7] Suif-Infrastructure: http://suif.stanford.edu/suif/

[8] Thomas Ball, "The Use of Control Flow and Control Dependence in Software Tools", *PhD Thesis, August 1993.*

[9] K.Cooper,T.Harvey and T.Waterman. "Building a Control-flow Graph from Scheduled Assembly Code",Technical Report, Department of Computer Science, Rice University, June 2002.

[10] Alfred V.Aho, Ravi Sethi and Jeffrey D. Ullman. Compilers Principles, Techniques and Tools, Addison-Wesley, 2006.

[11] WCET analysis tool http://www.absint.com/products.htm

[12] R.Heckmann, C.Ferdinand, "Worst-Case Execution Time Prediction by Static Program Analysis", *Proc. of the 18th Intl. Sym. on Par. and Dist. Processing, Apr. 2004.*

[13] Dongkun Shin, Seongsoo Lee, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications", *IEEE Design and Test of Computers,* Vol. 18, Issue 2, Mar-Apr. 2001.

[14] N.AbouGhazaleh, D.Mosse, B.Childers, R. Melhem, M.Craven, "Collaborative Operating System and Compiler Power Management for Real-Time Applications", *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium, 2003.*

[15] Y.H.Lu,"Comparing System Level Power Management Policies", *IEEE Design and Test of Computers,* vol. 18, pp. 10 – 19, 2001.

[16] V.Swaminathan, K.Chakrabarty, "Energy Conscious, Deterministic I/O Device Scheduling in Hard Real-Time Systems", *IEEE Trans. on CAD of Int. Ckts and Sys.*, vol. 22, No. 7, 2003.

[17] H.Cheng, S.Goddard, "Online Energy-Aware I/O Device Scheduling for Hard Real-Time Systems", *Proc. of Design, Automation and Test in Europe, Mar. 2006.*

[18] Y.H.Lu, L.Benini, G.D.Micheli,"Power-Aware Operating Systems for Interactive Systems", *IEEE Trans.on VLSI Systems,* Vol. 10, No. 2, 2002.

[19] L.Benini, A.Bogliolo, S.Cavallucci, B.Ricco, "Monitoring System Activity for OS-Directed Dynamic Power Management", *Proc. of the Intl. Symp. on Low Power Electronics and Design, Aug. 1998.*

[20] C.Gniady, A.Butt,Y.Hu, and Y.H.Lu. "Program Counter Based Prediction Techniques for Dynamic Power Management", *IEEE Transactions on Computers*, 55(6):641–658, June 2006.