The Slowdown or Race-to-idle Question: Workload-Aware Energy Optimization of SMT Multicore Platforms under Process Variation

Anup Das, Geoff V. Merrett and Bashir M. Al-Hashimi
School of ECS, University of Southampton, United Kingdom
Email: {a.k.das,gvm,bmah}@ecs.soton.ac.uk

Abstract—Two widely used approaches for reducing energy consumption in multithreaded workloads are slowdown (using DVFS) and race-to-idle. In this paper, we first demonstrate that most energy-efficient choice is dependent on (1) workload (memory bound, CPU bound etc.), (2) process variation and (3) support for Simultaneous Multithreading (SMT). We then propose an approach for mapping application threads on SMT multicore systems at run-time, to minimize energy consumption. The proposed approach interfaces with the OS and hardware performance counters to characterize application threads. This characterization captures the effect of process variation on execution time and identifies the break-even operating point, where one strategy (slowdown or race-to-idle) outperforms the other. Thread mapping is performed using these characterized data by iteratively collapsing application threads (SMT) followed by binary programming-based thread mapping. Finally, performance slack is exploited at run-time to select between slowdown and race-to-idle, based upon the break-even operating point calculated for each individual thread. This end-to-end approach is implemented as a run-time manager for the Linux OS and is validated across a range of high performance applications. Results demonstrate up to 13% energy reduction over all state-of-the-art approaches, with an average of 18% improvement over Linux.

I. INTRODUCTION

SMT-based multicore systems are emerging as the de facto platforms for achieving manycore performance with power efficiency using a limited number of multicore CPUs [1]. Earlier works on these platforms have focused primarily on thread mapping to improve performance [2]. These approaches are implemented as an OS kernel module with information from hardware performance counters. With strict thermal and power budgets, the focus is shifting towards power-aware thread mapping on multicore platforms (e.g. [3]). Most of these approaches use linear programming or heuristics to generate an energy minimum thread mapping considering single thread execution on a core at any given time (i.e., no SMT). As a result, there exists significant scope for further energy optimization if these approaches are used in SMT-based multicore systems. To address this, the technique presented in [4] uses a thread consolidation heuristic, replacing the OS default scheduler, for power-aware thread placement on chip multiprocessors.

As transistor geometry shrinks to sub-32nm scales, non-uniform gate-oxide thickness, random doping fluctuations and non-precise lithography cause large variability in processor microarchitecture, specifically affecting the threshold voltage $V_{th}$ and the effective length $L_{eff}$ of transistors. Process variation has substantial impact on two major parameters of a processor – the frequency it can attain and the leakage power it consumes. As multiple CPU cores are integrated on the same SoC, these cores can be expected to have variations both in frequency and power consumption [5]. Recently, studies have been conducted for process-variation aware thread mapping on multi-/many-core systems, to improve system performance and energy consumption (e.g. [6]). None of these approaches consider SMT-based multicore platforms, and additionally, all these variation-tolerant approaches scale down processor voltage and frequency to reduce energy consumption. As we demonstrate in this work, processor slowdown is not optimum to minimize energy consumption under all circumstances. For certain workload variations, it is beneficial to adopt a race-to-idle strategy i.e., executing the workload at the highest voltage-frequency, switching to an idle state upon completion [7].

To address this, we introduce an end-to-end approach for mapping application threads on SMT multicore systems at run-time, addressing process variation-aware energy optimization. The proposed run-time approach interfaces with the OS and hardware performance counters, to characterize an application’s threads, storing these statistics in a characterization table. This table is exploited to generate thread mapping decision using binary integer programming (BIP) and thread collapsing (SMT), iteratively. Finally, execution slack is exploited to select between slowdown and race-to-idle, utilizing the workload statistics. The remainder of this paper is organized as follows. Selection between slowdown and race-to-idle is discussed in Section II. The optimization problem is formulated in Section III. The iterative run-time approach is discussed in Section IV. Results are discussed in Section V and the conclusion in Section VI.

II. SLOWDOWN VS RACE-TO-IDLE

Figure 1(a) shows the energy ratio of slowdown vs race-to-idle for five single-threaded applications at four different frequencies. The break-even margin (energy ratio = 1) is shown in the figure as a red solid line. Results in this figure are interpreted as follows. If the energy ratio is $< 1$ at a particular frequency, it means it is energy-efficient to use slowdown; otherwise, it is more energy efficient to use race-to-idle. As seen in the figure, the race-to-idle strategy is more energy efficient than slowdown for all applications at 1.53 GHz. However, at 2.12 GHz, slowdown is more efficient. In most existing run-time approaches, the decision for slowdown or race-to-idle is typically taken before loading an application. Once selected, a control algorithm performs the desired action whenever there is slack in the application.

In our proposed approach we characterize an application to determine this break-even operating point. Based on the available slack we begin with scaling down the frequency until the break-even point. Upon reaching this point, we switch to race-to-idle. The scenario however, becomes more complicated when considering multithreaded workloads, in which case different threads can potentially have different break-even frequencies, and an overall selection has to be made for the application as a whole (see Algorithm 2). A second consideration is process variation, which influences the leakage power consumption, increasing or decreasing the energy ratio. The windows around these applications in Figure 1(a) highlight the maximum to minimum variation of the energy ratio. Figure 1(b) shows this variation for the whetstone application plotting the results with the nominal values for the parameters.
Energy Ratio 0.95 1.15 0.9 1.1 0.9 1.1 1 (E)

During this time, the energy consumption using race-to-idle cores. It is easy to relate the execution time of thread execution time (and the power consumption) for different variation is incorporated by distinguishing between a thread’s execution time. To evaluate the improvement of one strategy over the other, we define the following:

**Slowdown**: A strategy where voltage and frequency are scaled down to reduce energy.

**Race-to-Idle**: A strategy where a thread is executed at the highest operating condition i.e., (vN, fN), switching to idle state upon completion.

The energy consumption of thread i on core cj at reduced operating point (vI, fI) is ES D(i, j, l) = P I→l · t I→l. The total time taken by this thread to complete execution is t I→l. During this time, the energy consumption using race-to-idle strategy is E R23(i, j, l) = P I→l · t I→l + (t I→l − t I→l) · P I. In computing the energy consumption of race-to-idle strategy, the idle power consumption of the core is also taken into account for the extra time duration (second term in the equation). To evaluate the improvement of one strategy over the other, we define the thread-centric energy-ratio of thread i on core cj at operating point (vi, fi) as rE(i, j, l) = ES D(i, j, l)/ER23(i, j, l).

**TABLE I. NOTATIONS AND LEGENDS USED IN THIS PAPER**

\( (v_1, f_1) \) = Voltage-frequency pair of the platform \( 1 \leq l \leq N_i \)

\( c_1, \ldots, c_{N_i} \) = Cores of the platform

\( t_{i,l}^{v_1,f_1} \) = Execution time of thread i on core cj \( (v_1, f_1) \)

\( P_{i,l}^{v_1,f_1} \) = Average power consumption of thread i on core cj \( (v_1, f_1) \)

\( P_{i,l}^{v_1} \) = Idle power consumption of core cj

**III. PROBLEM FORMULATION**

**A. Energy Improvement: Slowdown vs Race-To-Idle**

Table I shows the notation used in this work. Process variation is incorporated by distinguishing between a thread’s execution time (and the power consumption) for different cores. It is easy to relate the execution time of thread i on core cj as \( t_{i,l}^{v_1,f_1} \geq t_{i,l}^{v_2,f_2} \). We define the following:

\[ \text{Slowdown:} \text{ A strategy where voltage and frequency are scaled down to reduce energy.} \]

\[ \text{Race-to-Idle:} \text{ A strategy where a thread is executed at the highest operating condition i.e., (vN, fN), switching to idle state upon completion.} \]

\[ \text{The energy consumption of thread i on core cj at reduced operating point (vI, fI) is } E_{SD}(i, j, l) = P_{i,l}^{v_1,f_1} \cdot t_{i,l}^{v_1,f_1}. \text{ The total time taken by this thread to complete execution is } t_{i,l}^{v_1,f_1}. \]

\[ \text{During this time, the energy consumption using race-to-idle strategy is } E_{R23}(i, j, l) = P_{i,l}^{v_1,f_1} \cdot t_{i,l}^{v_1,f_1} + (t_{i,l}^{v_1,f_1} - t_{i,l}^{v_1,f_1}) \cdot P_{i,l}^{v_1}. \text{ In computing the energy consumption of race-to-idle strategy, the idle power consumption of the core is also taken into account for the extra time duration (second term in the equation). To evaluate the improvement of one strategy over the other, we define the thread-centric energy-ratio of thread i on core cj at operating point (vI, fI) as } r_E(i, j, l) = E_{SD}(i, j, l)/E_{R23}(i, j, l). \]

**IV. AN ITERATIVE RUN-TIME APPROACH**

Figure 2 shows the proposed iterative approach for process variation-aware thread mapping, compared to existing approaches [6] and [8]. There are three steps involved – thread characterization, iterative SMT mapping, and energy optimization. The thread characterization step collects important performance statistics (including execution time) of
every thread on every core. The thread mapping step uses the characterization data to determine a mapping using binary integer programming (BIP). Two threads are identified in the mapping which results in the highest energy savings (using Equation 2) by running them together on a core (SMT). This step is identified as "Thread Collapsing". The collapsed thread is considered as a single thread and is executed on all cores to characterize it. Once this is completed, the BIP is re-executed with this new data and the process is repeated. The iterative approach continues as long as thread collapsing reduces energy consumption. Details of these steps are provided next.

A. Variation-Aware Thread Characterization

The average power consumption of thread $i$ on core $c_j$ can be written as $P_{l,i}^{j} = g(v_{l,i}, f_{l,i}, pmu)_{1}, pmu)_{2}, \ldots, pmu)_{M}$, where $(v_{l,i}, f_{l,i})$ is the voltage and frequency of operation of thread $i$ and $pmu$ is the reading of performance monitoring unit (PMU) registers $A$ [9]. Thread characterization is performed by executing these threads on all cores at highest frequency, collecting all necessary statistics. A two-dimensional characterization table is populated with these data.

B. BIP-Based Thread Mapping

We define a mapping variable $x_{i,j}$, where

$$x_{i,j} = \begin{cases} 1 & \text{if thread } T_i \text{ is mapped on core } c_j \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

with the following constraints

- A thread can only be mapped to a single core i.e., $\sum_j x_{i,j} = 1 \quad \forall i$
- The total execution time of all the threads mapped on a core must satisfy the deadline requirement i.e., $\sum_i x_{i,j} \cdot t_{i} \leq D \quad \forall j$

The objective is to minimize energy consumption i.e.,

$$\min E = \sum_{i,j} x_{i,j} \cdot P_{l,i}^{j} \cdot t_{i} \quad (4)$$

C. Energy-Aware Thread Collapsing

To identify two threads that result in the highest energy improvement upon collapsing, we predict the execution time of the collapsed thread using the approach proposed in [11]. Algorithm 1 provides the pseudo-code for the proposed thread collapsing approach. A core is identified with the highest number of expanded threads (line 1). The expanded threads on this core are put in an array $ThdArr$ (line 2). For every pair of threads of this array, the collapsed-mode execution time is calculated using [10] and the cross-thread energy ratio using Equation 2. If the energy ratio is greater than the maximum ratio computed thus far, the maximum value is updated. The algorithm terminates when all thread pairs are explored.

Algorithm 1 Thread Collapsing

Input: Thread mapping 
Output: Threads $a$ and $b$ that will be collapsed 
1: $c_j$ = A core where threads are expanded 
2: $ThdArr$ = Set of expanded threads on $c_j$ 
3: Initialize $r_{max} = 0$, $a = b = \emptyset$ 
4: for all $i, i' \in ThdArr$ do 
5: \quad $t_{i',j} = \text{Predict collapsed execution time using [10]}$ 
6: \quad Compute $r = r_{i,j}(i', j, N_l)$ 
7: \quad if $r > r_{max}$ then $r_{max} = r$, $a = i$ and $b = j$ 
8: end for 
9: Return $a, b$

D. Collapsed Thread Characterization

Once the threads to be collapsed are identified, the next step is to update the characterization table by replacing these threads with the collapsed ones. The collapsed thread is now considered as a single thread and is executed on all cores to collect the necessary statistics.

E. Energy Optimization: Slowdown vs Race-to-Idle

Algorithm 2 provides the pseudo-code, considering the generic case where frequency of the cores can be altered independently. The algorithm has two sections – frequency characterization (lines 1-5) followed by operating point selection (lines 6-14). For frequency characterization, the operating points of all cores are varied in lock-step from their minimum to maximum value. For every setting, the application is executed for an iteration, recording the execution time of all threads (including the collapsed ones). These data are stored in a two dimensional array $TArr$ corresponding to each thread-operating point pairs. After the characterization step, the operating point of each core is determined. For this, threads mapped to a core ($c_j$) are first stored in an array $TArr(j)$. The operating point $(v_{l,i}, f_{l,i})$ which results in the least positive slack is selected (line 8). The overall thread-centric energy improvement is determined (lines 10-12). If this is $< 1$ (implying slowdown has a lower energy consumption than race-to-idle), $(v_{l,i}, f_{l,i})$ is selected as the frequency of the core; else $(v_{N_l}, f_{N_l})$ is selected. At the end of this step, a voltage-frequency pair is selected for each core.

Algorithm 2 Slowdown vs Race-to-Idle

Input: Thread allocation, deadline $D$ 
Output: Core frequency selection 
1: for all $(v_{l,i}, f_{l,i})$ \quad $1 \leq l \leq N_l$ do 
2: Set $(v_{l,i}, f_{l,i})$ on all cores 
3: Execute an iteration of the application using the thread allocation 
4: Record execution time for each thread in $TArr$ 
5: end for 
6: for all $c_j$ \quad $1 \leq j \leq N_c$ do 
7: $TArr(j)$ = threads on core $c_j$ 
8: $(v_{l,i}, f_{l,i}) = \text{argmin} \quad D - \sum_{i,j} \text{exec time} \cdot TArr(i, k)$ 
9: Initialize $r = 1$ 
10: for all $\forall thd \ i \in TArr(j)$ do 
11: \quad $r = r \times r_{i,j}$ 
12: end for 
13: if $r < 1$ select $(v_{l,i}, f_{l,i})$ else select $(v_{N_l}, f_{N_l})$ 
14: end for

V. RESULTS AND VALIDATION

We validate our approach on an Nvidia Tegra multicore platform running Linux Kernel 3.10.24 with process variation modeled using [12]. A range of high performance applications are considered from the PARSEC and the SPLASH2 suites.

A. Slowdown and Race-to-Idle

Figure 3(a) plots the frequency selection for the proposed approach for five applications over 15 iterations. For dijkstra, basmaths and sha applications, the proposed approach switches to the highest frequency of 2.33 GHz after scaling down to a certain frequency (the break-even frequency). For other applications such as gsm and stringsearch, the proposed approach uses slowdown as this is more energy efficient than race-to-idle. The energy results (Figure 3(b)) confirm this frequency selection, showing that the proposed approach always selects the energy minimum strategy.
VI. CONCLUSION

An end-to-end approach is proposed for energy-aware mapping of application threads on a multicore platform, taking into account SMT and process variation. Application slack is exploited by selecting between race-to-idle and slowdown. The choice is guided by (1) application workload (CPU intensive, memory intensive, etc.), (2) process variation and (3) SMT. Experiments with high performance applications on a real platform, and using proven process variation models, demonstrate that the proposed approach improves energy consumption by up to 13%, while achieving similar performance as state-of-the-art approaches. Our continuing work considers energy optimization with multiple simultaneous applications.

ACKNOWLEDGMENT

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the PRiME Programme Grant EP/K034448/1 (www.prime-project.org). Experimental data used in this paper can be found at DOI: DOI:10.5258/SOTON/404445 (http://doi.org/10.5258/SOTON/404445).

REFERENCES