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

978-3-9815370-6-2/DATE16/

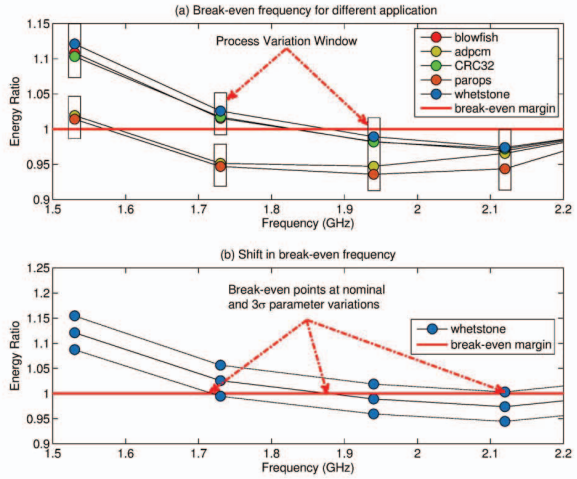

Fig. 1. Slowdown vs race-to-idle choice for different applications.

(V_{th} and L_{eff}) and the 3σ variations. As can be seen, process variation causes the break-even frequency to change (from no break-even point to one at 1.73 GHz). We determine the process variation-aware break-even frequency through thread characterization.

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 c_j as $t_i^{j\to 1} \geq t_i^{j\to 2} \geq \cdots t_i^{j\to N_l}$. 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., (v_{N_l}, f_{N_l}), switching to idle state upon completion.

The energy consumption of thread i on core c_j at reduced operating point (v_l, f_l) is $E_{SD}(i,j,l) = P_i^{j\to l} \cdot t_i^{j\to l}$. The total time taken by this thread to complete execution is $t_i^{j\to l}$. During this time, the energy consumption using race-to-idle strategy is $E_{R2I}(i,j,l) = P_i^{j\to N_l} \cdot t_i^{j\to N_l} + \left(t_i^{j\to l} - t_i^{j\to N_l}\right) \cdot P_I^j$. 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 c_j at operating point (v_l, f_l) as $r_s(i,j,l) = E_{SD}(i,j,l)/E_{R2I}(i,j,l)$.

(v_l, f_l)	= Voltage-frequency pair of the platform $\mid 1 \leq l \leq N_l$
$c_1, \cdots, c_{N_c}$	= Cores of the platform
$t_i^{j\to l}$	= Execution time of thread i on core c_j @ (v_l, f_l)
$P_i^{j\to l}$	= Average power consumption of thread i on core c_j @ (v_l, f_l)
P_I^j	= Idle power consumption of core c_j

TABLE I. Notations and legends used in this paper

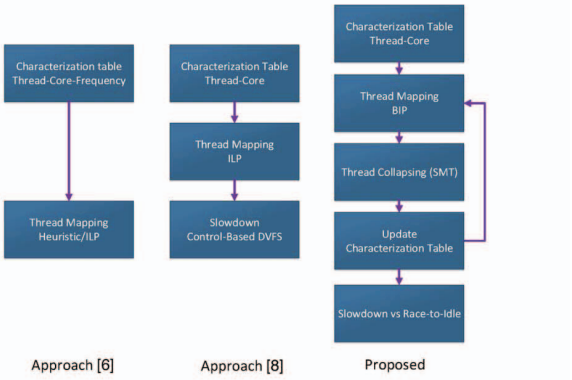

Fig. 2. Traditional and iterative run-time thread mapping approaches.

This equation simplifies to

$$r_s(i,j,l) = \frac{\alpha_i^{j\to l} \cdot s_i^{j\to l}}{\alpha_i^{j\to N_l} + s_i^{j\to l} - 1} \tag{1}$$

where $\alpha_i^{j\to l} = P_i^{j\to l}/P_I^j$ is the spread of dynamic and idle power for thread i on core c_j operating at (v_l, f_l) and $s_i^{j\to l} = t_i^{j\to l}/t_i^{j\to N_l}$ is its execution time slowdown factor.

B. Energy Improvement: Expanded vs Collapsed Thread

`Expanded Mode:` Two threads are executed in expanded mode if they are executed one after another.

`Collapsed Mode:` Two threads are executed in collapsed mode if they are executed simultaneously on a core.

We consider two threads i and i' executed on core c_j operating at (v_l, f_l). We are interested in finding the break-even point (if one exists) where one of the above modes of operation is better than the other. The total energy consumed in executing these threads in expanded mode is $E_{ex}(i,i',j,l) = P_i^{j\to l} \cdot t_i^{j\to l} + P_{i'}^{j\to l} \cdot t_{i'}^{j\to l}$. Let the power consumption and execution time of the two threads in collapsed mode of operation be indicated by the subscript i_i'. The energy consumption in the collapsed mode of operation is given by $E_{co}(i,i',j,l) = P_{i\_i'}^{j\to l} \cdot t_{i\_i'}^{j\to l} + \left(t_i^{j\to l} + t_{i'}^{j\to l} - t_{i\_i'}^{j\to l}\right) \cdot P_I^j$, where we have also accounted for the idle power for the extra duration (the second part). To define the break-even point, we introduce the *cross-thread energy-ratio* of threads i and i' on core c_j operating at $(v_l.f_l)$, defined as $r_c(i,i'j,l) = E_{ex}(i,i',j,l)/E_{co}(i,i',j,l)$. The above equation simplifies to:

$$r_c(i,i',j,l) = \frac{\alpha_i^{j\to l} \cdot m_i^{j\to l} + \alpha_{i'}^{j\to l} \cdot m_{i'}^{j\to l}}{\alpha_{i\_i'}^{j\to l} + m_i^{j\to l} + m_{i'}^{j\to l} - 1} \tag{2}$$

where $m_i^{j\to l} = t_i^{j\to l}/t_{i\_i'}^{j\to l}$ is the SMT slowdown of execution time. Equation 2 can be generalized to any number of threads. However, for the proposed approach we restrict it to two threads at a time. This simplifies the energy improvement equation and provides fine control over thread collapsing.

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

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: $t^j_{i\_i'}$ = Predict collapsed execution time using [10]
6: Compute $r = r_c(i, i', j, N_l)$
7: **if** $r > r_{max}$ **then** $r_{max} = r, a = i$ and $b = j$
8: **end for**
9: Return a, b

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 $P_i^{j\rightarrow l} = g(v_l, f_l, pmu1_i^j, pmu2_i^j, \cdots, pmuM_i^j)$, where (v_l, f_l) is the voltage and frequency of operation of thread i and $pmuA_i^j$ 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., $\sum_j x_{i,j} = 1 \ \ \forall i$
- The total execution time of all the threads mapped on a core must satisfy the deadline requirement i.e., $\sum_i x_{i,j} \cdot t_i^j \leq D \ \ \forall j$

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

$$\min \ E = \sum_{i,j} x_{i,j} \cdot P_i^j \cdot t_i^j \quad (4)$$

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 2 Slowdown vs Race-to-Idle

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

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 `ThdTimeArr` 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, f_l) 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, f_l) 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.

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, basicmaths 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.

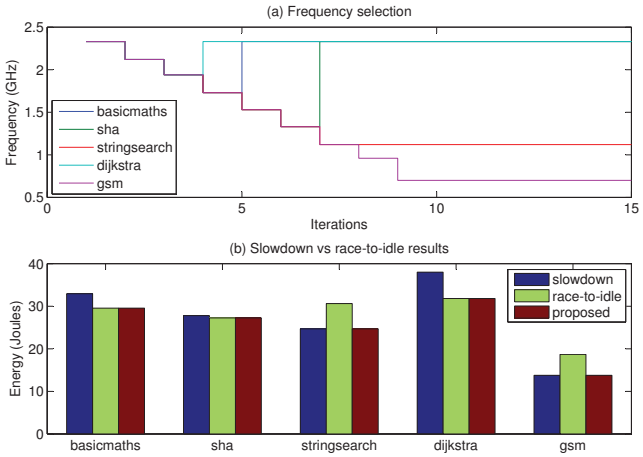

Fig. 3. Slowdown vs race-to-idle comparison.

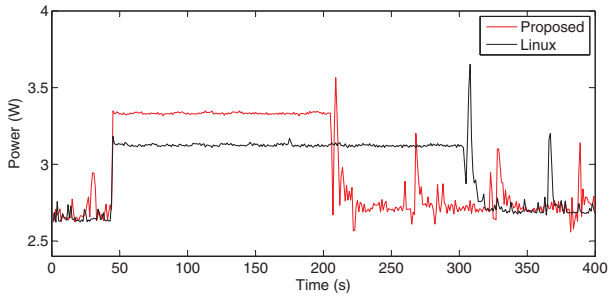

Fig. 4. Power comparison of Linux and proposed approach.

B. Energy and Execution Time Results

Figure 4 plots the real power traces obtained from an Agilent n6705b DC power analyzer while executing the raytrace application. Results are compared for one iteration of the application comparing the thread mapping generated using the proposed approach with that obtained using Linux's default mapping. As can be seen from the figure, the proposed approach uses race-to-idle strategy for this application, resulting in higher power consumption than Linux but much lower execution time. Overall, the energy consumption (measured by the area in the plot) is much lower (510 J) as compared to the slowdown strategy (800 J) implemented by Linux (ondemand governor). On average for all applications, the improvement using the proposed approach is 18% as compared to Linux.

Figure 5 plots the energy and execution time results for five applications, comparing the results of the proposed approach with those obtained using existing techniques of [6] and [4]. There are a few trends to observe from this figure. First the technique of [6] has both a higher execution time and a higher energy consumption compared to the other two approaches. This is because the technique of [6] does not consider SMT, and therefore leaves a significant scope for energy optimization. The execution time of [4] and the proposed approach are similar. However, in terms of energy consumption, the proposed approach consumes less energy. For some applications, such as streamcluster, the improvement is 13%. On average, the improvement is 7%.

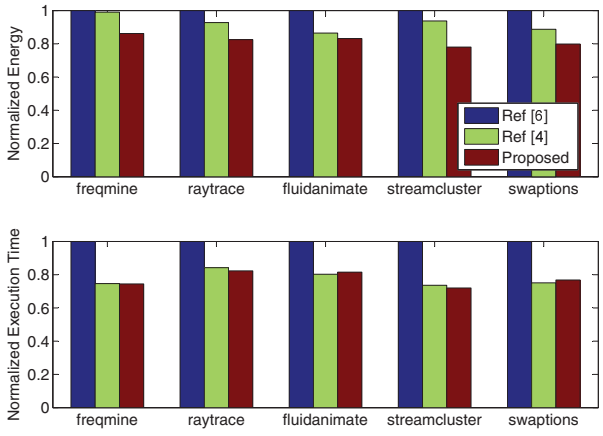

Fig. 5. Energy and execution time results.

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).

References

[1] B. Sinharoy, J. Van Norstrand *et al.*, "IBM POWER8 processor core microarchitecture," *IBM Journal of Research and Development*, 2015.

[2] L. Porter *et al.*, "Making the most of SMT in HPC: System-and application-level perspectives," *ACM TACO*, 2015.

[3] V. Petrucci *et al.*, "Energy-efficient thread assignment optimization for heterogeneous multicore systems," *ACM TECS*, 2015.

[4] A. Vega *et al.*, "SMT-centric power-aware thread placement in chip multiprocessors," in *PACT*, 2013.

[5] K. Agarwal *et al.*, "Characterizing process variation in nanometer CMOS," in *DAC*, 2007.

[6] F. Fraternali *et al.*, "Quantifying the impact of variability on the energy efficiency for a next-generation ultra-green supercomputer," in *ISLPED*, 2014.

[7] S. Albers and A. Antoniadis, "Race to idle: new algorithms for speed scaling with a sleep state," *ACM Transactions on Algorithms*, 2014.

[8] M. Kandemir *et al.*, "Dynamic thread and data mapping for noc based cmps," in *DAC*, 2009.

[9] M. J. Walker *et al.*, "Run-time power estimation for mobile ad embedded asymmetric multi-core cpus." HiPEAC EEHCO, 2015.

[10] R. Garibotti *et al.*, "Simultaneous multithreading support in embedded distributed memory mpsocs," in *DAC*, 2013.

[11] Y. Zhang, M. A. Laurenzano *et al.*, "SMiTe: Precise QoS prediction on real-system SMT processors to improve utilization in warehouse scale computers," in *International Symposium on Microarchitecture*, 2014.

[12] B. Raghunathan *et al.*, "Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors," in *DATE*, 2013.