Estimating the Worst-Case Energy Consumption of Embedded Software

Ramkumar Jayaseelan Tulika Mitra Xianfeng Li
School of Computing
National University of Singapore
{ramkumar,tulika,lixianfe}@comp.nus.edu.sg

Abstract

The evolution of battery technology is not being able to keep up with the increasing performance demand of mobile embedded systems. Therefore, battery life has become an important design constraint. As battery-operated embedded devices are deployed in mission critical systems, designers should ensure that the energy constraints are satisfied in addition to the timing constraints — the battery should not drain out before a critical task completes execution. Giving these guarantees requires the knowledge of the worst-case execution time and energy of a task. Significant progress has been made in estimating the worst-case execution time through static analysis. In contrast, existing energy estimation techniques use average-case execution profile of a program and as such cannot guarantee the satisfiability of energy constraints. In this paper, we present a static analysis technique to estimate the worst-case energy consumption of a task on complex micro-architectures. Estimating a bound on energy is non-trivial as it is unsafe to assume any direct correlation with the bound on execution time. Experimental evaluation with a number of benchmark programs indicates the accuracy of our worst-case energy consumption estimates.

1 Introduction

The proliferation of battery-operated embedded devices has made energy consumption one of the key design constraints. Increasingly, mobile devices are demanding improved functionality and higher performance. Unfortunately, the evolution of battery technology is not being able to keep up with the performance requirements. Therefore, significant research effort has focused on conserving energy to prolong battery life. All these techniques are targeted towards the average-case energy consumption of a task. On the other hand, designers of mission critical systems, operating on limited battery life, have to ensure that both the timing and the energy constraints are satisfied under all possible scenarios — the battery should never drain out before a task completes its execution. Conventional schedulability analysis techniques can guarantee the satisfiability of timing constraints for hard real-time systems. One of the key inputs required for this schedulability analysis is the worst-case execution time (WCET) of the tasks. A decade of research in static timing analysis has solved the WCET estimation problem to a large extent. The related problem of estimating the **worst-case energy consumption (WCEC)** remains largely unexplored even though it is considered highly important [21] especially for mobile devices. In this paper, we take the first step towards understanding and estimating the worst-case energy consumption of a task executing on a particular processor for all possible inputs.

Estimating WCEC is in particular important for remotely deployed embedded systems such as nodes of wireless sensor networks that depend on environmental energy (e.g., solar power) for battery recharge. The sensor nodes should run perpetually on ambient energy without manual recharging or replacement of batteries. Often such sensor networks are deployed for mission critical applications (e.g., defense applications) and must satisfy timing and energy constraints. Recently, scheduling algorithms [10] have been proposed for distributed sensor networks that take into account the spatio-temporal profile of the available energy resources at the different nodes. These algorithms can exploit accurate timing analysis results for sensor network nodes [17]. But they assume energy consumption of a task corresponding to some "representative" inputs. As a result, they cannot guarantee that the task, when scheduled on a particular sensor node, will complete its execution before the battery drains out. The knowledge of WCEC is crucial in this scenario. Similarly, reward-based scheduling algorithms [19] that attempt to satisfy both timing and energy constraints can benefit from the WCEC estimates. It is also important for battery-operated embedded systems catering to a mix of critical and non-critical tasks. Given a critical task τ and a non-critical task τ', the operating system may not schedule τ' if the summation of the WCEC values of τ and τ' exceeds the remaining battery power.

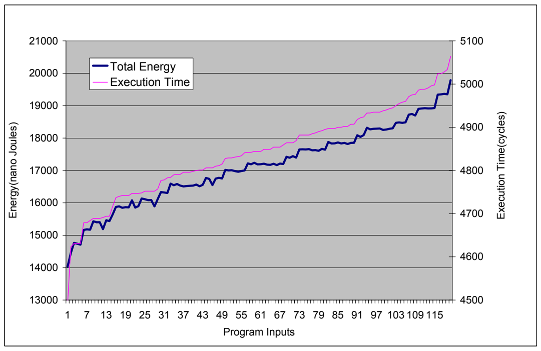

Figure 1. *Variations in execution time versus total energy for different inputs of insertion sort.*

Clearly, simulating the execution of a task for all possible inputs so as to bound its energy consumption is not feasible. Also, the complexity of current micro-architectures makes it highly improbable that the critical execution path can be determined without the help of analytical methods. Thus, we propose a static analysis technique to estimate safe and tight bounds for the worst-case energy consumption.

A natural question that may arise is the possibility of using the WCET path to compute a bound on the worst-case energy consumption. As $energy = average\ power \times execution\ time$, this may seem like a viable solution and one that can exploit the extensive research in WCET analysis in a direct fashion. Unfortunately, the path corresponding to the WCET may not coincide with the path consuming maximum energy. Figure 1 plots the variation in execution time and total energy corresponding to all possible inputs of an insertion sort program (for a 5-element array)[1]. The inputs for this program are plotted along the x-axis according to increasing execution time. There are a large number of points in this plot where the energy consumption decreases with increasing execution time. This happens because dynamic energy (due to switching activity in the circuit) need not necessarily have a correlation with the execution time. So, the input that leads to WCET may not be identical to the input that leads to WCEC.

How can we then compute the worst-case energy for a code fragment? Traditional power simulators, such as Wattch [2] and Simple Power [23], perform cycle-by-cycle power estimation and then add them up to obtain total energy consumption. Clearly, we cannot use cycle-accurate estimation to compute worst-case energy bound as it would essentially require us to simulate all possible scenarios (which is too expensive). The other method [22, 20] is to use fixed per-instruction energy; but it fails to capture the effects of cache miss and branch prediction. Instead, our WCEC analysis technique is based on the key observation that the energy consumption of a program can be separated out into the following time-dependent and time-independent components.

Instruction-specific energy The energy that can be attributed to a particular instruction (e.g., energy consumed due to the execution of the instruction in the ALU, cache miss, etc.). Instruction-specific energy does not have any relation with the execution time.

Pipeline-specific energy The energy consumed in the various hardware components (clock network power, leakage power, switch-off power etc.) that cannot be attributed to any particular instruction. Pipeline-specific energy is roughly proportional to the execution time.

Thus, we avoid cycle-accurate simulation by estimating the two energy components separately. Pipeline-specific energy estimation can exploit the knowledge of WCET. However, as we will see later, the definition of switch-off power and clock network power makes the energy analysis much more involved — we cannot simply multiply the WCET by a constant power factor. Moreover cache misses and overlap among basic blocks due to pipelining and branch prediction adds significant complexity to our analysis. In summary, the contributions of our work are as follows

- To the best of our knowledge, this is the first work that attempts to derive bound on worst-case energy consumption of a task through static analysis.
- We propose a safe but tight worst-case energy consumption estimation method. Experimental evaluation shows that our analysis derives quite accurate estimates for a large number of benchmarks.

Organization The rest of the paper is organized as follows. Section 2 briefly introduces the power-related terminologies used in the paper. We review the previous work in energy estimation and timing analysis in Section 3. Sections 4 and 5 present our core technique to estimate the worst-case energy consumption of a program. We evaluate the accuracy of our estimation method through experimental evaluation in Section 6. Section 7 concludes the paper and outlines possible future directions.

2 Background

Power and energy are two different terms that are often used interchangeably as long as the context is clear. For battery life, however, the important metric is energy rather than power. The energy consumption of a task running on a processor is defined as $Energy = P \times t$, where P is the average power and t is the execution time. Energy is measured in Joules whereas power is measured in Watts (Joules/sec).

[1]Details of the experimental setup are given in Section 6.

Power consumption consists of two main components: dynamic power and leakage power $P = P_{dynamic} + P_{leakage}$.

Dynamic power is caused by the charging and discharging of the capacitive load on each gate's output due to switching activity. It is defined as $P_{dynamic} = \frac{1}{2} A V_{dd}^2 C f$ where A is the switching activity, V_{dd} is the supply voltage, C is the capacitance and f is the clock frequency. For a given processor architecture, V_{dd} and f are constants. The capacitance value for each individual component of the processor can be derived through RC-equivalent circuit modeling [2].

Switching activity A is dependent on the particular program being executed. For circuits that charge and discharge every cycle, such as double-ended array bitlines, an activity factor of 1.0 can be used. However, for other circuits (e.g., single-ended bitlines, internal cells of decoders, pipeline latches etc.), an accurate estimation of the activity factor requires examination of the actual data values. It is difficult, if not impossible, to estimate the activity factors through static analysis. Therefore, we conservatively assume an activity factor of 1.0 (i.e., maximum switching) for each active processor component. However, most processors have some form of clock gating (which we do model) that disables unused components and effectively lowers the activity factor. Moreover, previous research [22] has shown that the switching activity (so called "circuit-state effect") does not have significant impact on total power consumption.

Modern processors employ clock gating to save power. This involves switching off clock signals to the idle components so that they do not consume dynamic power in the unused cycles. We assume three different clock gating options following the model in [2].

Simple clock gating: This simplest gating style assumes that peak power will be consumed by a component if there is at least one access in a given cycle, and zero power otherwise. For example, peak power will be consumed in a multi-ported register file even if only one port is accessed.

Ideal clock gating: This gating style is most aggressive in that a multi-ported structure consumes power proportional to the number of ports accessed in a given cycle.

Realistic clock gating: Realistic clock gating is similar to ideal clock gating except that idle units/ports dissipate 10% of the peak power. This is a more realistic modeling as it may be impossible to totally shut-off power to an idle hardware unit/port. We refer to the power consumed in the idle cycles as **switch-off power**.

Clock distribution network consumes a significant fraction of the total energy. Without clock gating, **clock power** is independent of the characteristics of the applications. However, clock gating results in power savings in the clock distribution network. Whenever the components in a portion of the chip are idle, the clock network in that portion of the chip can be disabled reducing clock power.

Leakage power captures the power lost from the leakage current irrespective of switching activity. In this work, we use the leakage power model proposed in [24]: $P_{leakage} = V_{dd} \times N \times k_d \times I_{leakage}$, where V_{dd} is the supply voltage and N is the number of transistors. $I_{leakage}$ is a constant specifying the leakage current corresponding to a particular process technology. k_d is an empirically determined design parameter obtained through SPICE simulation corresponding to a particular device.

3 Related Work

In this section, we review related work in two different areas: architectural-level power/energy estimation techniques and static analysis techniques to estimate the worst-case execution time (WCET).

To the best of our knowledge this is the first work that attempts to estimate the worst-case energy consumption of a program. Many researchers have proposed methods to estimate the average-case power/energy consumption. Power consumption in a processor can be estimated at various levels; see [11] for a detailed survey. Low-level power estimation techniques are suitable to evaluate circuit-level techniques for saving power. Architectural-level power estimation techniques can be broadly classified into two categories: *cycle-accurate power simulators* and *instruction-level energy estimations*.

Cycle-accurate power simulators, such as Wattch [2] and SimplePower [23], are used to evaluate micro-architectural and compiler-based techniques to save power. Wattch has two main components: the parametrized power models and the architectural simulator. It works in association with SimpleScalar [1], a cycle-accurate micro-architectural simulator. The parameterized power models are used to estimate the power consumed per access for each component. The architectural simulator is used to determine the usage counts of the various components. The usage counts are multiplied by the power per access to obtain total energy.

Instruction-level power estimation methods have been presented in [22, 20]. In [22], a fixed energy cost is associated with each instruction. [20], on the other hand, only distinguishes between power consumption of different classes of instructions. Instruction-level energy estimation techniques are quite accurate for simple processor architectures. However, in the presence of complex architectural features, such as cache, pipeline, and speculation, instruction-level energy estimation is not sufficient. The key difference between our approach and instruction-level energy estimation techniques is that we have to be inherently conservative to derive a bound on the worst-case energy consumption.

Even though we are not aware on any work on estimating worst-case energy consumption, static analysis techniques to estimate worst-case execution time (WCET) is a wellresearched area [21]. Current research on WCET analysis takes the effect of micro-architectures into consideration. These include cache modeling [15, 5], pipeline modeling [4, 7, 9, 12] and branch prediction modeling [3, 13]. In fact, commercial WCET analysis tools that can model complex processor architectures (e.g., Motorola ColdFire, PowerPC 755) are currently available in the market [6].

4 Estimation of Worst Case Energy

The starting point of our analysis is the control flow graph of the program. The first step of our analysis is to estimate an upper bound on the energy consumption of each individual basic block. Once these bounds are known, we can estimate the worst case energy of the entire program.

4.1 Processor Model

For an architecture without pipeline, cache, and other performance enhancing features, there is no variation in the energy consumption of a basic block. Thus, a variety of techniques can be employed to compute the energy consumption of a basic block including cycle-accurate simulation [2, 23] and software-level power estimation [22, 20]. Estimating a tight bound on the energy consumption of a basic block gets difficult as the complexity of the micro-architecture increases. However, in the embedded domain, many recent processors employ out-of-order pipelines, cache and branch prediction; examples include Motorola MPC 7410, PowerPC 755, PowerPC 440GP, AMD-K6 E and NEC VR5500 MIPS. Sensor network nodes are increasingly employing complex processors (e.g., Intel XScale PXA255 processor based Stargate sensor gateway) and even out-of-order execution [18].

In this section, we first assume a simplified processor model that has out-of-order pipeline but perfect instruction cache and branch prediction. Integration of the effects of cache miss and branch misprediction are discussed in the next section. The processor model we use is a slightly modified version of the SimpleScalar [1] `sim-outorder` simulator processor model. It has a standard 5-stage pipeline consisting of Instruction Fetch (`IF`), Instruction Decode & Dispatch (`ID`), Instruction Execute (`EX`), Write Back (`WB`) and Commit (`CM`). Instruction fetch, decode, and commit occur in program order. However, instructions can proceed out-of-order in execute and write-back stages based on dependency and resource contention. A central structure in this pipeline is a circular buffer, called the re-order buffer (ROB). Instructions remain in the ROB from the time they are dispatched to the time they are committed. After decoding, instructions are dispatched to ROB *in program order*. But instructions can be issued from the ROB to execution units *out-of-order*.

4.2 Energy Estimation for a Basic Block

Our goal here is to estimate a tight upper bound on the total energy consumption $energy_{BB}$ of a basic block BB through static analysis. From the discussion in Section 2

$$\begin{aligned} energy_{BB} = \quad & dynamic_{BB} + switchoff_{BB} \\ & + leakage_{BB} + clock_{BB} \end{aligned} \tag{1}$$

where $dynamic_{BB}$ is the instruction-specific energy component, i.e., the energy consumed due to switching activity as an instruction goes through the pipeline stages. $switchoff_{BB}$, $leakage_{BB}$, and $clock_{BB}$ are defined as the energy consumed due to the switch-off power, leakage power, and clock power, respectively during $wcet_{BB}$ where $wcet_{BB}$ is the worst case execution time of the basic block BB. The worst case execution time($wcet_{BB}$) is estimated using the static analysis technique in [12]. Now we describe how to define bounds for each individual energy component.

4.2.1 Instruction-specific Energy

The instruction-specific energy of a basic block is the dynamic power consumed due to the switching activity generated by the instructions in that basic block.

$$dynamic_{BB} = \sum_{instr \in BB} dynamic_{instr} \tag{2}$$

where $dynamic_{instr}$ is the dynamic power consumed by an instruction $instr$. Now, let us analyze the energy consumed by an instruction as it travels through the pipeline. *The estimation model below assumes ideal/realistic clock gating (see Section 2).* Both these clock gating styles assume that dynamic power consumed in multi-ported structure is proportional to the number of accessed ports. Later we will modify it to support simple clock gating.

Fetch and decode: The energy consumed here is due to fetch, decode and instruction cache access. As we assume perfect instruction cache, we do not need to model cache misses. Refinement of the analysis to account for instruction cache misses is discussed in the next section.

Register access: The energy consumed for the register file access due to reads/writes can vary from one class of instructions to another. Assuming ideal/realistic clock gating, the energy consumption in the register file for an instruction is proportional to the number of register operands.

Branch prediction: The energy consumption is fixed as we assume perfect branch prediction here. In the next section, we discuss modeling of branch misprediction.

Wakeup logic: When an operation produces a result, the wakeup logic is responsible to make the dependent instructions ready and the result is written onto the result bus. An instruction places the tag of the result on the wakeup logic and the actual result on the result bus exactly once and the corresponding energy can be easily accounted for. The enery consumed in the wakeup logic is proportional to the number of output operands.

Selection logic: Selection logic is interesting from the point of view of energy consumption. The selection logic is responsible to select an instruction to execute from a pool of ready instructions. Unlike the other components discussed earlier, an instruction may access the selection logic more than once. This is because an instruction can request for a specific functional unit and the request might not be granted in which case it makes a request in the next cycle. However, we cannot accurately determine the number of times an instruction would access the selection logic. Therefore, *we conservatively assume that the selection logic is accessed every cycle.*

Functional units: The energy consumed by an instruction in the execution stage depends on the functional unit it uses and its latency. For variable latency instructions, we assume the maximum energy consumption. We assume a perfect data cache and hence energy consumption for load/store units is also constant and equal to the energy consumption of the data cache.

Now, Equation 2 corresponding to dynamic energy consumed in a basic block BB is redefined as

$$\begin{aligned} dynamic_{BB} &= selection\_power_{cycle} \times wcet_{BB} \\ &\quad + \sum_{instr \,\in\, BB} dynamic_{instr} \end{aligned} \tag{3}$$

where $selection\_power_{cycle}$ is a constant defining the power consumed in the selection logic per cycle. $wcet_{BB}$ is the worst case execution time of BB. Note that $dynamic_{instr}$ is redefined as the power consumed by $instr$ in all the pipeline stages except for selection logic.

Modifications for simple clock gating style The previous discussion assumes ideal/realistic clock gating where the energy consumption in a multi-ported structure is proportional to the number of accesses per cycle. In contrast, for simple clock gating, a multi-ported structure consumes peak power even if there is only one access in that cycle. Therefore, we need to modify the dynamic energy component corresponding to multi-ported hardware structures, namely caches, register file etc. Corresponding to each multi-ported hardware structure C, we compute the total number of accesses to that structure by the instructions in basic block BB called $access_{BB}(C)$. As we cannot determine the distribution of these accesses over the execution period of BB, we simply assume that full power will be consumed in component C for $min(access_{BB}(C), wcet_{BB})$ cycles. Note that such an assumption is conservative and hence the estimated dynamic energy of basic block ($dynamic_{BB}$) is an upper bound.

4.2.2 Pipeline-specific Energy

As mentioned before, pipeline-specific energy consists of three components: switch-off energy, clock-energy and leakage energy. All three energy components are influenced by the execution time of the basic block.

Switch-off Energy The switch-off energy refers to the power consumed in an idle unit when it is disabled through clock gating. Switch-off energy is zero for ideal clock gating and simple clock gating styles (see Section 2). However, we need to model switch-off power for realistic clock gating. Let $access_{BB}(C)$ be the total number of accesses to a component C by the instructions in basic block BB. Let $ports(C)$ be the maximum number of allowed accesses/ports for component C per cycle. Then, we define switch-off energy for component C in basic block BB as

$$\begin{aligned} switchoff_{BB}(C) = &\left(wcet_{BB} - \frac{access_{BB}(C)}{ports(C)}\right) \\ &\times full\_power_{cycle}(C) \times 10\% \end{aligned} \tag{4}$$

where $full\_power_{cycle}(C)$ is the full power consumption per cycle for component C. The switch-off energy corresponding to a basic block can now be defined as

$$switchoff_{BB} = \sum_{C \in components} switchoff_{BB}(C) \tag{5}$$

where $components$ is the set of all hardware components.

Clock Network Energy In order to estimate the energy consumed in the clock network, we should take clock gating into account. As we evaluate the accuracy of our estimation technique against Wattch [2], we use their approximation for clock gating. Our modeling of conditional clocking power can be easily adapted to accurately reflect the underlying clock distribution scheme.

$$clock_{BB} = non\_gated\_clock_{BB} \times \left(\frac{circuit_{BB}}{non\_gated\_circuit_{BB}}\right) \tag{6}$$

where $non\_gated\_clock_{BB}$ is the clock energy without gating and can be defined as

$$non\_gated\_clock_{BB} = clock\_power_{cycle} \times wcet_{BB} \tag{7}$$

where $clock\_power_{cycle}$ is the peak power consumed per cycle in the clock network. $circuit_{BB}$ is defined as the power consumed in all the components except clock network in the presence of clock gating. That is,

$$circuit_{BB} = dynamic_{BB} + switchoff_{BB} + leakage_{BB} \tag{8}$$

$non\_gated\_circuit_{BB}$, on the other hand, is the power consumed in all the components except clock network in the absence of clock gating. It is simply defined as

$$non\_gated\_circuit_{BB} = circuit\_power_{cycle} \times wcet_{BB} \tag{9}$$

$circuit\_power_{cycle}$ is a constant defining peak dynamic plus leakage power per cycle excluding the clock network.

Leakage Energy The leakage energy is simply define as $leakage_{BB} = P_{leakage} \times wcet_{BB}$ where $P_{leakage}$ is the power lost per processor cycle from the leakage current regardless of the circuit activity. This quantity, as defined in Section 2, is a constant given a processor architecture. $wcet_{BB}$ is as usual the worst-case execution time of BB.

4.3 Estimation for the whole program

Given the energy bounds for the basic blocks, we can now estimate the WCEC of a program using an Integer Linear Programming (ILP) formulation. The ILP formulation is similar to the one originally proposed by Li and Malik [15] to estimate the WCET of a program. We replace the execution time of the basic blocks with the corresponding energy consumptions. We briefly describe the ILP formulation here for the sake of completeness.

The input to the ILP formulation is the control flow graph (CFG) of the program. The vertices of the CFG are the basic blocks with their corresponding energy bounds. An edge $B_i \rightarrow B_j$ denotes the flow of control from basic block B_i to B_j. We assume that the CFG has a unique start node (B_{start}) and a unique end node (B_{end}) such that all program paths originate at the start node and terminate at the end node. For programs with procedures and functions (recursive or otherwise), we create a separate copy of the CFG of a procedure P for every distinct call site of P. Each call of P transfers control to its corresponding copy.

Flow constraints and loop bounds Let $count_{B_i}$ denote the number of times basic block B_i is executed, and $count_{B_i \rightarrow B_j}$ denote the number of times control flows through the edge $B_i \rightarrow B_j$. As inflow equals outflow for each basic block (except for the start and end nodes)

$$count_{B_i} = \sum_{B_j} count_{B_j \rightarrow B_i} = \sum_{B_j} count_{B_i \rightarrow B_j}$$

As the start and end blocks are executed exactly once:

$$count_{B_{start}} = \sum_{B_i} count_{B_{start} \rightarrow B_i} = 1$$

$$count_{B_{end}} = \sum_{B_i} count_{B_i \rightarrow B_{end}} = 1$$

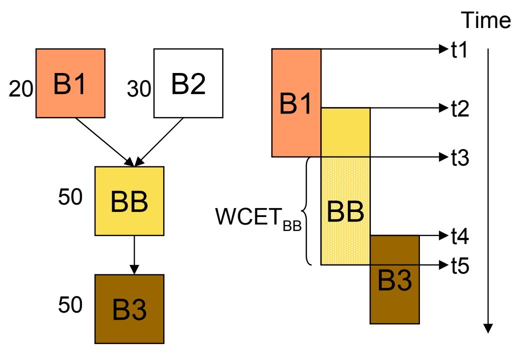

Figure 2. *Illustration of overlap.*

Of course, we need bounds on the maximum number of iterations for loops and maximum depth of invocations for recursive procedures. These bounds can be user provided, or can be computed off-line for certain programs [8].

Objective function Let $energy_{B_i}$ be the upper bound on the energy consumption of a basic block B_i. Then the total energy consumption of the program is given by

$$Total\ \ energy = \sum_{i=1}^{N} energy_{B_i} \times count_{B_i} \tag{10}$$

where the summation is taken over all the basic blocks in the program. The worst-case energy consumption of the program can be derived by maximizing the objective function under the flow constraints through an ILP solver.

4.4 Execution overlap among basic blocks

So far, for the simplicity of exposition, we have not discussed the issue of overlap among the basic blocks due to the pipelined nature of execution. A major difficulty in estimating WCEC, however, arises from the overlapped execution of basic blocks. Let us illustrate the problem with a simple example. Figure 2 shows a small portion of the CFG. Suppose we are interested in estimating the energy bound for basic block BB. The annotation for each basic block indicates the maximum execution count. This is just to show that the execution counts of overlapped basic blocks can be different. As the objective function (defined by Equation 10) multiplies each $energy_{BB}$ with its execution count $count_{BB}$,we cannot arbitrarily transfer energy between overlapping basic blocks. Clearly, instruction specific energy of BB should be estimated based on only the energy consumption of its instructions. However we cannot take such a simplistic view for pipeline specific energy. Pipeline-specific energy depends critically on $wcet_{BB}$.

If we define $wcet_{BB}$ without considering the overlap, i.e., $wcet_{BB} = t_5 - t_2$, then it results in excessive overestimation of the pipeline-specific energy values as the time intervals $t_3 - t_2$ and $t_5 - t_4$ are accounted for multiple times. To avoid this, we can re-define the execution time of BB as the time difference between the completion of execution of the predecessor (B_1 in our example) and the completion of execution of BB, i.e., $wcet_{BB} = t_5 - t_3$. Of course, if BB has multiple predecessors then we need to estimate $wcet_{BB}$ for each predecessor and then take the maximum value among them.

This definition of execution time, however, cannot be used to estimate the pipeline-specific energy of BB in a straightforward fashion. This is because, switch-off energy (for realistic clock gating) and thus clock network energy depend on the idle cycles for hardware ports/units. As we are looking for worst-case energy, we need to estimate an upper bound on idle cycles. Idle cycles estimation (see Equation 4) requires an estimate of $access_{BB}(C)$, which is defined as the total number of accesses to a component C by the instructions in basic block BB. Now, with the new definition of $wcet_{BB}$ as the interval $t_5 - t_3$, not all these accesses fall within $wcet_{BB}$ and we run the risk of underestimating idle cycles. To avoid this problem, we replace in Equation 4 $access_{BB}(C)$ with $access_{BB}^{WCET_{BB}}(C)$, which is defined as the total number of accesses to a component C by the instructions in basic block BB that are guaranteed to occur within $wcet_{BB}$. Note that we do not change the definition of $access_{BB}(C)$ for calculating the dynamic energy for simple clock gating. This is because we are interested on the upper bound on the number of accesses in that case.

To estimate the accesses according to this new definition, we take a closer look at the WCET estimation technique for each basic block which is based on [12]. Basically, it is an interval-based iterative technique that estimates earliest/latest start and completion time for the different pipeline stages of each instruction in the basic block. Due to space constraints, we refer interested readers to [12] for more details. Now, let t_3 be the latest commit time of the last instruction of the predecessor node B_1 and t_5 be the earliest commit time of the last instruction of BB. Then, for each pipeline stage of the different instructions in BB, we check whether its earliest or latest start time falls within the interval $t_5 - t_3$. If the answer is yes, then the accesses corresponding to that pipeline stage are guaranteed to occur within $wcet_{BB}$ and are included in $access_{BB}^{WCET_{BB}}(C)$. We now estimate the pipeline-specific energy w.r.t. each of BB's predecessors and take the maximum value.

5 Integrating cache and branch prediction

In the previous section, we computed worst-case energy by assuming perfect instruction cache and branch prediction. In this section, we integrate the modeling of cache miss and branch misprediction. The modeling of these micro-architectural features exploits our previous work in the context of WCET analysis [14]. Our previous work presents an integrated ILP-based modeling where program path analysis as well as branch prediction/ instruction cache behavior are formulated as linear constraints; an ILP solver is used to maximize the objective function denoting program's execution time. In this paper, we directly use the ILP formulation of [14] to capture flow analysis, instruction cache and branch prediction effects. But we modify the estimation of the constants denoting the energy values of basic blocks. Due to space consideration, we will only mention the modifications required for energy analysis.

The basic idea is to define different scenarios for a basic block corresponding to cache miss and branch misprediction. If these scenarios are defined suitably, then we can estimate a constant that bounds the energy consumption of a basic block corresponding to each scenario. Finally, the execution frequencies of these scenarios are defined as ILP variables and are bounded by additional linear constraints.

Scenarios corresponding to cache misses are defined as follows. Given a cache configuration, a basic block BB can be partitioned into a fixed number of memory blocks, with instructions in each memory block being mapped to the same cache block (cache accesses of instructions other than the first one in a memory block are always hits). A **cache scenario** of BB is defined as a mapping of hit or miss to each of the memory blocks of BB. We now compute upper bounds on energy of BB w.r.t. each of the possible cache scenarios. Basically, for each cache scenario, we need to add the dynamic energy due to cache misses defined as

$$mem\_energy_{BB}^{\omega} = miss_{BB}^{\omega} \times access\_energy \qquad (11)$$

where $mem\_energy_{BB}^{\omega}$ is the main memory energy for BB corresponding to cache scenario ω, $miss_{BB}^{\omega}$ is the number of cache misses in BB corresponding to cache scenario ω, and $access\_energy$ is a constant defining the energy consumption per main memory access. Of course, we need to re-estimate the pipeline-specific energy for each cache scenario by taking into account the WCET corresponding to that scenario.

Similarly, the scenarios for branch prediction are defined as the two branch outcomes (correct prediction and misprediction) corresponding to each of the predecessor basic blocks. Branch misprediction results in additional dynamic energy consumption due to the the execution of additional instructions along the mispredicted path. In addition, misprediction may increase the WCET of a basic block resulting in additional pipeline-specific energy. We follow the modeling proposed in [14] that estimates the WCET for each scenario. This takes care of the additional pipeline-specific energy due to misprediction.

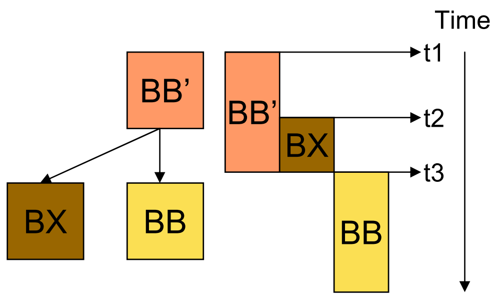

Figure 3. *Illustration of branch misprediction.*

We estimate the additional instruction-specific energy due to the execution of speculative instructions as follows. Let BB be a basic block with BB' as the predecessor (see Figure 3). If there is a misprediction for the control flow $BB' \rightarrow BB$, then instructions along the basic block BX will be fetched and execution. The executions along this mispredicted path will continue till the commit of the branch in BB'. Let t_3 be the latest commit time of the mispredicted branch in BB'. For each of the pipeline stages of the instructions along the mispredicted path (i.e., BX), we check if its earliest start time is before t_3. If the answer is yes, then the dynamic energy for that pipeline stage is added to the branch misprediction energy of BB. In this fashion, we estimate the WCEC of a basic block BB corresponding to all possible scenarios, where a scenario consists of a preceding basic block BB', correct/wrong prediction of the conditional branch in BB' and the cache scenario of BB.

We now redefine our ILP formulation to integrate our analysis of pipelining, instruction cache and branch prediction. Let $B_1, \ldots, B_N$ be the set of basic blocks of the program whose WCEC we are estimating. Now the execution of B_i is associated with its cache scenarios and the prediction of its preceding branch. We denotes the set of possible cache scenarios at B_i as Ω_i. Considering the possible cache scenarios and correct/wrong prediction of the preceding branch for a basic block, the ILP objective function denoting a program's total energy is now written as follows.

$$Energy = \sum_{i=1}^{N} \sum_{j \rightarrow i} \sum_{\omega \in \Omega_i} energy_{j \rightarrow i}^{c,\omega} * count_{j \rightarrow i}^{c,\omega} + energy_{j \rightarrow i}^{m,\omega} * count_{j \rightarrow i}^{m,\omega} \qquad (12)$$

where $energy_{j \rightarrow i}^{c,\omega}$ is the WCEC of B_i executed under the following scenario: (1) B_i is reached from a preceding block B_j, (2) the branch prediction at the end of B_j is correct or B_j does not have a conditional branch, and (3) B_i is executed under a cache scenario $\omega \in \Omega_i$. Also, $count_{j \rightarrow i}^{c,\omega}$ is the number of times that B_i is executed under this context. Similarly, $energy_{j \rightarrow i}^{m,\omega}$ is the WCEC of B_i executed under the following scenario: (1) B_i is reached from a preceding block B_j, (2) the branch at the end of B_j is mispredicted, and (3) B_i is executed under a cache scenario $\omega \in \Omega_i$. Again, $count_{j \rightarrow i}^{m,\omega}$ is the number of times that B_i is executed under this context.

We have already estimated $energy_{j \rightarrow i}^{c,\omega}$ and $energy_{j \rightarrow i}^{m,\omega}$ as constants. We use the constraints developed in [14] to bound the ILP variables $count_{j \rightarrow i}^{c,\omega}$ and $count_{j \rightarrow i}^{m,\omega}$. Finally, we maximize the objective function under these constraints.

6 Experimental Results

In this section we evaluate the accuracy of our estimation technique using some benchmarks commonly used for WCET analysis. [2] We use the SimpleScalar [1] framework for our experiments. We model a processor with out-of-order pipeline, instruction cache and branch prediction. Our estimator can be parameterized w.r.t to the cache configuration, branch predictor configuration, the number of entries in the instruction window, the latency of the functional units etc. The processor model for these experiments has 8-entry instruction window, 4-entry fetch queue and the following functional units: a single-cycle integer ALU, an integer multiplier with $1 \sim 4$ cycle latency, a floating-point adder with $1 \sim 2$ cycle latency and a floating-point multiplier with $1 \sim 12$ cycle latency. We assume a single-cycle load/store unit (i.e., perfect data cache). We model a 4KB instruction cache with 4-way associativity, 32-byte block size and LRU replacement policy. The cache hit latency is one cycle and cache miss latency is 10 cycles. We assume a gshare branch predictor with 2-bit branch history register and 16-entry branch prediction table. The clock frequency of the processor is assumed to be 600 MHz and the supply voltage is 2.5 V.

We use the parameterized power models of Wattch [2], a micro-architectural level power simulator, to estimate the energy consumed per access for the hardware components. Our power models differ from Wattch in certain aspects. First, Wattch does not model the main memory energy corresponding to cache misses. We use the energy per access for a 256MB DDR RAM obtained from the micron system power calculator [16]. Second, Wattch does not distinguish among different integer and floating-point operations as far as the energy consumption is concerned. For example, it uses the same energy value for integer addition and multiplication. We fix this problem in our power models. Finally, we model leakage power for array based structures such as register files and cache [24] but Wattch does not.

Given the binary executable of a program, we first construct the control flow graph. Procedure calls are handled by replicating the control flow graph associated with the procedure for each call and adding edges appropriately at the call and return sites. Next, our analyzer estimates the energy at

[2]The insertion sort program here sorts 100-element array as opposed to 5-element array used for the plots in Figure 1.

basic block level using the power models. Finally, it formulates the objective function and constraints for instruction cache, branch prediction as well as the flow constraints for the ILP solver. We use CPLEX, a commercial ILP solver to obtain the estimated WCEC by maximizing the objective function under the constraints.

Table 1 presents the accuracy of our WCEC estimation technique for three different clock gating styles (see Section 2). *Est* represents the WCEC value returned by our estimation technique. For comparison purposes, we also need the actual WCEC (*Actual*) where $Est \geq Actual$. Unfortunately, we cannot find the actual WCEC for most benchmarks due to the large number of possible inputs. Instead, we use human guidance to select certain program inputs which we hope will lead a value that is close to the actual WCEC. We then use Wattch (with suitably modified power models for leakage and main memory energy) to simulate the program with these selected inputs and report the maximum observed energy. This maximum energy value obtained through simulation of a limited number of program inputs is called Observed WCEC (*Obs*). Clearly, *Obs* represents a lower bound of the actual WCEC. Therefore, $Est \geq Actual \geq Obs$. That is, *the accuracy of our estimation technique is at least as good as the results shown in Table 1, but could possibly look even better had we known the actual WCEC* .

Table 1 shows that the estimated WCEC value is quite close to the observed WCEC value for all the benchmarks. Ideal clock gating assumes zero switch-off energy for all idle units/ports. That is why it has the lowest energy value among the three clock gating styles. Simple clock gating is less aggressive in that it considers a unit idle only if there is no access for any port of the unit. Realistic clock gating assumes idle units/ports consume 10% of the peak power and hence it has the highest energy values. Notice that our estimation for ideal clock gating is also more accurate than that of realistic, simple clock gating. In case of realistic clock gating, a unit/port consumes switch-off energy during idle cycles. As idle cycles are estimated from the WCET, any over-estimation in the WCET results in over-estimation of switch-off energy. For simple clock gating style, the source of inaccuracy is different. In this case, idle units do not consume any switch-off energy. However, for multi-ported units, we now require the distribution of the accesses to that unit over the execution period in order to identify idle cycles for that unit. Unfortunately, we cannot determine this distribution accurately so we conservatively assume that every access occurs in a different clock cycle

Another source of inaccuracy is due to the fact that we only use minimal flow constraints in our ILP formulation and these flow constraints do not take into account infeasible path information. The execution counts of the basic blocks returned by the ILP solver are often higher than the actual execution counts during simulations. That is, the execution counts returned by the ILP solver may not match with the execution profile of any feasible program path.

Benchmark	WCET $\times$ AvgPower (μJ)	Observed (μJ)
isort	**489.92**	**525.88**
fft	12106.49	10260.86
fdct	138.20	105.57
ludcmp	131.76	119.33
matsum	**972.03**	**1154.31**
minver	93.612	80.80
bsearch	3.84	3.07
des	724.05	643.75
matmult	178.12	166.88
qsort	54.79	43.73
qurt	23.80	17.65

Table 2. *Problem of WCEC estimation using WCET.*

As discussed in Section 1, estimating the WCEC of a task as $WCET \times AvgPower$ may lead to under-estimation, i.e., it is unsafe. This is shown in Table 2 where for the highlighted benchmarks, the $Est\ WCET \times AvgPower$ is less than even the observed worst case energy consumption.

Finally, we note that our estimation technique is quite fast. For estimating the WCEC corresponding to the realistic clock gating style (as it is the most time consuming), it takes only $0.15 \sim 2.88$ seconds to formulate the ILP problems for the benchmark programs. ILP solver is even faster and completes under 1.8 seconds for all the benchmarks. All the experiments have been performed on a Pentium IV 1.3 GHz PC with 1 GB of memory.

7 Conclusions

We have presented a static analysis technique to estimate the worst-case energy consumption of a program on a complex processor architecture with out-of-order pipeline, instruction cache, and branch prediction. The worst-case energy consumption can help designers give power guarantees for battery-operated embedded devices just as designers of conventional real-time systems provide timing guarantees. Experimental results indicate that our estimation is quite accurate. In future, we would like to validate our estimation results against commercial embedded processors. We would also like to explore the possibility of extending our technique to provide thermal guarantees.

8 Acknowledgments

This work was partially supported by NUS research grant R252-000-171-112.

Benchmark	**Simple Clock Gating**			**Ideal Clock Gating**			**Realistic Clock Gating**		
	Est(μJ)	**Obs(μJ)**	**Ratio**	**Est(μJ)**	**Obs(μJ)**	**Ratio**	**Est(μJ)**	**Obs(μJ)**	**Ratio**
isort	524.95	455.94	1.15	468.85	422.76	1.11	596.93	525.88	1.14
fft	11057.50	9185.39	1.20	9600.66	8586.49	1.12	13631.21	10260.86	1.33
fdct	99.31	88.78	1.11	89.92	83.63	1.08	121.65	105.57	1.15
ludcmp	115.39	100.32	1.15	98.75	92.77	1.06	139.75	119.33	1.17
matsum	1227.37	994.11	1.23	1012.83	929.74	1.09	1397.72	1154.31	1.21
minver	74.91	64.15	1.17	63.66	59.61	1.07	90.95	80.80	1.13
bsearch	3.51	3.07	1.14	2.537	2.40	1.06	3.81	3.07	1.24
des	613.16	553.74	1.10	546.415	518.22	1.05	715.58	643.75	1.11
matmult	172.39	136.93	1.26	149.706	132.08	1.13	212.94	166.88	1.28
qsort	39.50	33.84	1.17	34.90	31.16	1.12	49.84	43.73	1.14
qurt	16.36	12.97	1.26	13.98	11.91	1.17	21.95	17.65	1.24

Table 1. *Accuracy of our worst-case energy estimation technique.*

References

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2), 2002.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2000.

[3] A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Journal of Real time Systems*, 18(2/3), 2000.

[4] J. Engblom. *Processor Pipelines and Static Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Sweden, 2002.

[5] C. Ferdinand and R. Wilhelm. Fast and Efficient Cache Behavior Prediction for Real-Time Systems. *Real-Time Systems*, 17(2/3), 1999.

[6] A. A. I. GmbH. aiT: Worst case execution time analyzer, 2004. `http://www.absint.com/ait/`.

[7] C. Healy, R. Arnold, F. Mueller, D. Whalley, and M. Harmon. Bounding pipeline and instruction cache performance. *IEEE Transactions on Computers*, 48(1):53–70, 1999.

[8] C. Healy et al. Supporting timing analysis by automatic bounding of loop iterations. *Journal of Real-Time Systems*, 18(2/3), 2000.

[9] R. Heckmann et al. The Influence of Processor Architecture on the Design and the Results of WCET Tools. *Proceedings of the IEEE*, 91(7), 2003.

[10] A. Kansal, D. Potter, and M. B. Srivastava. Performance aware tasking for environmentally powered sensor networks. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2004.

[11] P. Landman. High-level power estimation. In *Proceedings of the 1996 International Symposium on Low Power Electronics and Design*, 1996.

[12] X. Li, T. Mitra, and A. Roychoudhury. Modeling out-of-order processors for software timing analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2004.

[13] X. Li, T. Mitra, and A. Roychoudhury. Modeling control speculation for timing analysis. *Real-Time Systems Journal*, 29(1), 2005.

[14] X. Li, A. Roychoudhury, and T. Mitra. Modeling out-of-order processors for WCET analysis. Technical Report TRC9/05, National University of Singapore, September 2005.

[15] Y.-T. S. Li, S. Malik, and A. Wolfe. Performance estimation of embedded software with instruction cache modeling. *ACM Transaction on Design Automation of Electronic Systems (TODAES)*, 4(3), 1999.

[16] Micron. The micron system-power calculator. `http://www.micron.com/products/dram/syscalc.html`.

[17] S. Mohan, F. Mueller, D. Whalley, and C. Healy. Timing analysis for sensor network nodes of the Atmega processor family. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2005.

[18] L. Nazhandali et al. A second-generation sensor network processor with application-driven memory optimizations and out-of-order execution. In *CASES*, 2005.

[19] C. Rusu, R. Melhem, and D. Mossé. Maximizing rewards for real-time applications with energy constraints. *ACM Transactions on Embedded Computing Systems*, 2(4), 2003.

[20] A. Sinha and A. P. Chandrakasan. Jouletrack: A web based tool for software energy profiling. In *Proceedings of the Design Automation Conference (DAC)*, 2001.

[21] L. Thiele and R. Wilhelm. Design for timing predictability. *Real-Time Systems Journal*, 28(2/3), 2004.

[22] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions of VLSI Systems*, 2(4), 1994.

[23] W. Ye et al. The design and use of simplepower: A cycle-accurate energy estimation tool. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2000.

[24] Y. Zhang et al. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical Report CS-2003-05, University of Virginia, 2003.