Application-Driven Processor Design Exploration for Power-Performance Trade-off Analysis*

Diana Marculescu, Anoop Iyer
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890
{dianam,aiyer}@ece.cmu.edu

Abstract - *This paper presents an efficient design exploration environment for high-end core processors. The heart of the proposed design exploration framework is a two-level simulation engine that combines detailed simulation for critical portions of the code with fast profiling for the rest. Our two-level simulation methodology relies on the inherent clustered structure of application programs and is completely general and applicable to any microarchitectural power/performance simulation engine. The proposed simulation methodology is 3-17X faster, while being sufficiently accurate (within 5%) when compared to the fully detailed simulator. The design exploration environment is able to vary different microarchitectural configurations and find the optimal one as far as energyxdelay product is concerned in a matter of minutes. The parameters that are found to affect drastically the core processor power/performance metrics are issue width, instruction window size, and pipeline depth, along with correlated clock frequency. For very high-end configurations for which balanced pipelining may not be possible, opportunities for running faster stages at lower voltage exist. In such cases, by using up to 3 voltage levels, the energyxdelay product is reduced by 23-30% when compared to the single voltage implementation.*

1 Introduction

In recent years, power dissipation has become a critical design concern not only for designers of battery powered or wireless electronics, but also in the case of high-performance microprocessors, multimedia and digital signal processors or high-speed networking. While it is generally agreed that tools for power estimation and optimization do exist at circuit, gate, register-transfer or behavioral levels [1], less has been done in the area of power analysis or optimization at microarchitecture, architecture or system-level [2]. Having tools that are able to quantify the effect of different performance or power optimization schemes, for a piece of code running on a given processor, is of extreme importance for computer architects and compiler engineers who can characterize different architecture styles not only in terms of their performance, but also in terms of the corresponding energy efficiency. This may also enable the fine tuning of any existing energy/performance trade-offs.

Relevant previous work comes from several different areas such as software power modeling for general purpose, embedded or Digital Signal Processing (DSP) applications, compilation techniques for power optimized software, pipeline scheduling for low power, and energy efficient memory systems. Specifically, [3] proposes a *per-instruction base power model* that can be used to find an aggregate power estimate for a sequence of instructions. The *inter-instruction overhead* due to changes in circuit state are also taken into account, but their effect was found to be negligible. The approach presented in [4] targets instruction scheduling to reduce circuit state overhead. The proposed *cold scheduling* technique reduces the switching activity in the control path by reordering the instructions based on the inter-instruction energy overhead. For the special case of real-time systems, an approach for reducing energy via scheduling, while still meeting the deadlines, has been presented in [5]. In [6], the case of DSP applications is addressed. There, while the same type of models as in [3] have been developed, the inter-instruction effects turn out to be much more prevalent, thus making possible to develop instruction scheduling techniques that target power minimization. In [7] a more efficient model with space complexity linear in the instruction set size has been presented.

More recently, [8] and [9] present two more advanced microarchitectural power simulators. In [8], a very accurate parameterized power simulator (within 10% when compared to three different high-end microprocessors) is presented, as well as some interesting trade-offs between energy and performance under varying microarchitecture settings. In [9], the case of datapath dominated architectures is considered, as well as an analysis of the impact of compiler optimizations and memory design on power consumption values. A cycle-accurate power simulation environment has been presented in [10]. There, for the case of the SA-1100, a cycle-accurate power modeling methodology is developed. It cannot, however, model complex, superscalar processors with out-of-order execution.

In the area of energy optimization, the authors of [11] present an architectural enhancement to reduce the extra work or energy in the pipeline of a superscalar processor due to mispredicted branches, without significant loss in performance. In [12] a technique for reducing the average power consumption for the pipeline structure is presented. Other approaches target techniques for energy efficient memory systems, such as the use of selective cache ways [13], filter-caches [14] or code compression [15]. Several approaches in the most recent literature on power efficient architecture point out that different microarchitectural settings do provide different values for energy per instruction [16] or energyxdelay product [8]. It has also been observed that there exists a wide variation from one application to another, as well as between different parts of the same application both in terms of the inherent parallelism and in terms of the necessary resources to sustain a given performance level.

In this paper we address the problem of *efficient design exploration for low power* of modern processors. Specifically, we show that the power-optimal architectural configuration of the processor depends on *issue width*, *instruction window size* as well as *pipeline depth*. Processor design exploration has been addressed previously, but only considering the effect of issue width on the power-performance trade-off curve [16,17], or the effect of variable cache configurations on system power [18,19]. The difference in pipeline stage latencies has been explored for lowering the power requirements at system level, but for a specialized communication Myrinet GAM pipeline in a dynamic voltage scaling environment [20].

Our proposed design exploration methodology is up to *two orders* of magnitude (*8X* on average) *faster* and within *5% accurate* when compared to detailed microarchitectural simulators like SimpleScalar [21]. The estimation engine relies on a *two-level simulation methodology*: for critical parts of the code, an accurate, lower-level (but slow) simulation engine is invoked, whereas for non-critical parts of the application program, a fast, high-level, but less accurate simulation is performed. Critical parts of the code are identified using *hotspot detection* and *sampling* which is a novel and completely general paradigm, applicable to any detailed simulation environment. In addition to what other power simulators propose (e.g., *Wattch* [8]), in our power characterization, we account for data dependency for cycle-accurate estimates, as well as for

*This research was supported in part by NSF Career Award CCR-0084479.

parameterizable global clock power modeling.

Our design exploration framework is able to explore many possible microarchitectural configurations in a matter of minutes and find the best one in terms of power/performance trade-off. In addition to the proposed fast simulation engine, the proposed design exploration environment relies on latency analysis of different microarchitectural configurations (*Cacti* [22-24]). As the experimental results show: (i) for *complete* power/performance characterization, *issue width* is insufficient and has to be considered together with *instruction window* size and *pipeline depth*; and (ii) for very high-end configurations, *balanced pipelining* may not be possible, thus allowing for opportunities for running faster stages at lower voltage (and thus lower power). Savings of up to 23-30% in energyxdelay product can be achieved by using up to three different voltage levels.

To characterize the quality (in terms of power and performance) of different configurations, we rely on a few metrics of interest. As pointed out in [22], when characterizing the performance of modern processors, the *CPI* (Cycles per Instruction) or *IPC* (Instructions per Cycle, 1/*CPI*) is only one of two parameters that needs to be considered, the second one being the actual cycle time. Thus, the product $CPI*T_{cycle}$ is a more accurate measure for characterizing the performance of modern processors. In the case of power consumption, most researchers have concentrated on estimating or optimizing *energy per committed instruction* (*EPI*) or *energy per cycle* (*EPC*) [8,9,16]. While in the case of embedded computer systems with tight power budgets some performance may be sacrificed for lowering the power consumption, in the case of high performance processors this is not desirable and solutions that jointly address the problem of low power and high performance are needed. To this end, we propose the *energy delay product per committed instruction* (*EDPPI*) defined as $EPI*CPI*T_{cycle}$ as a measure that characterizes both the performance and power efficiency of a given architecture. Such a measure can identify microarchitectural configurations that keep the power consumption to a minimum, without significantly affecting the performance. In addition to classical metrics (such as *EPC* and *EPI*), we use this measure to assess the efficiency of our power optimization technique and to compare different configurations as far as power consumption is concerned.

The paper is organized as follows: Section 2 presents our assumptions, whereas in Section 3 we present our proposed processor design exploration environment. Section 4 presents the two-level microarchitecture simulation engine. Section 5 describes the details of the proposed design exploration environment and Section 6 shows our experimental results for the proposed technique applied on a subset of SpecInt95 benchmarks. We conclude with some final remarks and comments on the proposed approach.

2 Assumptions

In what follows, we consider a typical superscalar, out-of-order configuration, based on the reservation station model (Fig.1). This structure is used in modern processors like Pentium Pro and PowerPC 604. The main difference between this structure and the one used in other processors (like MIPS R10000, DEC Alpha 21264, HP PA-8000) is that the reorder buffer holds speculative values and the register file holds only committed, non-speculative data, whereas for the second case, both speculative and non-speculative values are in the register file. However, the wake-up, select and bypass logic are common to both types of architectures and, as pointed out in [22], their complexity increases significantly with increasing issue widths and window sizes.

We note that increasing issue widths have to go hand in hand with increasing instruction window sizes to provide significant performance gains. Thus, we argue that *complexity* (and thus, *power requirements)* of today's processors have to be characterized in terms of *issue width* (that is, number of instructions fetched, dispatched and executed in parallel), *instruction window size* (that is, the window of instructions that are dynamically reordered and scheduled for achieving higher parallelism), as well as *pipeline depth*, which is directly related to the operating clock frequency.

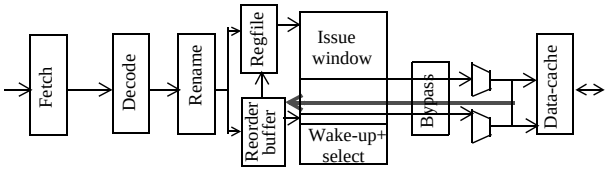

Fig.1 The reservation station model

As part of our design exploration framework, we rely on cycle-accurate simulation to get the performance and power consumption metrics, as well as techniques for speeding up simulation so that power-performance metrics are preserved with sufficient accuracy, while the simulation time is drastically reduced.

For our design exploration environment, we will assume that the parameters that affect the power-performance metrics of a given core processor are the *issue width*, *instruction window size* and *pipeline depth*. It is clear that pipelining depth is related to the clock rate (and thus the performance and power consumption of the core). However, it is also true that issue width and instruction window size play an important role, especially for higher end configurations, since they actually affect the number of resources used in the processor under consideration (datapath modules, memory ports, etc.). For example, for a 4-way superscalar processor which is able to fetch, issue and commit 4 instructions every clock cycle, there should be at least 2 integer ALUs, one floating-point ALU and at least one memory port available so that the datapath doesn't create any bottlenecks in case the code is highly parallel. On the other hand, issue width and instruction window size are inter-related one to another. More parallelism can be extracted by having a wider window size, and thus more bandwidth is needed for fetching, issuing and committing instructions. Finally, pipeline depth is determined by the way stages are balanced and by the inherent latency of different core modules.

In our design exploration environment we will concentrate on the effect of the above three parameters and we will assume that the instruction and data caches are fixed.

3 Processor design exploration

Today's superscalar, out-of-order processors pack a lot of complexity and functionality on the same die. Hence, design exploration for finding high performance or power efficient configurations is not an easy task. As shown in previous work [16-20], some of the factors that have a major impact on the power/ performance of a given processor are issue width, cache configuration, etc. However, as shown in [8], the issue window impacts strongly the power cost of a typical superscalar, out-of-order processor. To the best of our knowledge, the *issue width* (and corresponding number of functional units), *instruction window size*, as well as the *pipeline depth* have not been considered *simultaneously* as parameters in a design exploration environment.

Our proposed design exploration environment follows the flow in Fig.2. At the heart of the exploration framework is a fast microarchitectural simulator (**estimate_metrics**) that provides sufficiently accurate estimates for the metric of interest. Depending on the designer's needs, this metric can be one of: *CPI*, $CPI*T_{cycle}$, *EPI*, *EDPPI*, according to whether a high performance or a joint high-performance and energy-efficient organization is sought. As shown in Fig.2, the exploration is performed for a set of benchmarks *B*, a set of possible issue widths *I*, instruction window sizes *W* and a number of possible voltage levels *N*. For each pair (*issue width*, *instruction window size*), the stage latencies are estimated. If a balanced pipelined design is sought, the pipeline is further refined to account for this and only one voltage levels is assumed for the entire design. Otherwise, depending on the latencies of different stages, up to *N* different voltages are assigned to different modules such that performance constraints are maintained and the slowest stage dictates the operating clock frequency.

```
design_explore (B, I, W, N)
for each benchmark BN in B{
    for IW in I = (IW_1, IW_2,...,IW_n)
      for WS in W = (WS_1, WS_2,...,WS_m)
        estimate_stage_latencies (IW, WS);
        if (balanced_pipeline) {
          balance_stages (IW, WS);
          estimate_metrics (BN, IW, WS, 1);
        }
        else
          estimate_metrics (BN, IW, WS, N);
}
```

Fig.2 The design exploration environment

As it can be seen, the estimation engine is repeatedly invoked for varying sets of parameters. Thus, it is important to have a very fast, yet sufficiently accurate simulation engine (**estimate_metrics**). In the sequel, we show one possible solution to achieving these two, apparently contradictory, requirements.

4 Efficient microarchitectural power simulation

In this section we present the simulation methodology that is at the core of the entire design exploration framework. For a design exploration environment to be able to explore many possible design configurations in a short period of time, it has to rely either on a smart methodology to prune the design space, or on a fast, yet sufficiently accurate estimation tool for the metrics of interest. Our proposed design space exploration framework relies on the second approach.

The crux of our estimation speed-up methodology relies on a *two-level simulation methodology*: for critical parts of the code, an accurate, lower-level (but slow) simulation engine is invoked, whereas for non-critical parts of the application program, a fast, high-level, but less accurate simulation is performed. Following the principle "*make the common case accurate*," ideal candidates for critical sections that should be modeled accurately are those pieces of code where the application spends a lot of time in, which have been elsewhere called *hotspots* [25].

Example: Consider the collection of basic blocks[1] in Fig.3, where edges correspond to conditional branches and the weight of each edge is proportional to the number of times that direction of the branch is visited. *Hotspots* are collections of basic blocks that closely communicate one to another, but are unlikely to transition to a basic block outside of that collection. In Fig.3, basic blocks 1-4 and 5-9 are part of two different hotspots that communicate infrequently one to another.

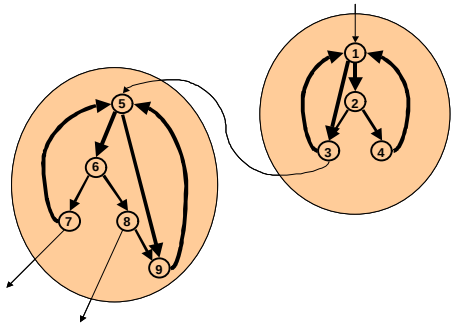

Fig.3 An example of two hotspots

As it will be seen later, these hotspots satisfy nice locality properties not only temporally, but also in terms of the behavior of the metrics that characterize power efficiency and performance. Temporal locality, as well as high probability of reusing internal variables [33] make hotspots attractive candidates for *sampling* metrics of interest over a fixed *sampling window*, after a *warm-up period* that would take care of any transient regimes. Estimated metrics obtained via sampling can be later reused when the exact same code is run again. Although different, such an approach is similar in some ways to power estimation techniques for hardware IPs using hierarchical sequence compaction [34] or stratified random sampling [35]. In addition, the relative sequencing of basic blocks is preserved (as in [36]) and the use of warm-up period ensures that overlapping of traces [37] is not necessary. This is in contrast with synthetically constructing traces for evaluating performance [38] and power consumption [39].

We should point out, however, that if used naïvely, the hotspot concept could become useless from an efficiency point of view. If detailed simulation is to be used for hotspots and they account for most of the execution time of the application program (70%-99% of the execution time, as we shall see later), then no significant savings in simulation time can be achieved. To speed-up the simulation time inside the hotspots and achieve the goal of "*making the common case fast*," we propose to use *sampling* of power and performance metrics until a given level of accuracy is achieved. This is supported by the fact that while being in a hotspot, both power consumption (expressed as *EPC* or *Energy per Cycle*) and performance (expressed as *IPC* or *Instructions Per Cycle*) achieve their stationary values within a short period of time, relative to the dynamic duration of the hotspot. As our experimental evidence shows, the steady-state behavior is achieved in less than 5% of the hotspot dynamic duration, thus providing significant opportunities for simulation speed-up, with minimal accuracy loss.

Fig.4 shows how the two-level simulation engine is organized. During detailed simulation, all performance and related power metrics are collected for cycle-accurate modeling. When a hotspot is detected, detailed analysis is continued for the entire duration of the sampling period. When sampling is done, the simulator is switched to basic profiling that only keeps track of the control flow of the application. Whenever the code exits the hotspot, detailed simulation is started again. This way, the error of estimation is conservatively bounded by the sampling error within the hotspots. Performing detailed simulation outside the hotspots ensures that the estimates are still accurate for benchmarks with low temporal locality (e.g., less than 60% time spent in hotspots).

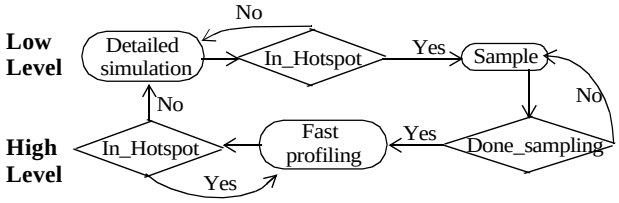

Fig.4 The two-level simulation engine

To complete our two-level simulation engine, we need a reliable and sufficiently detailed (albeit, slow) power/performance simulator as well as a rough (fast) profiler to keep track of where we are in the code. To this end, we use SimpleScalar [22] as the main engine for detailed performance estimation. SimpleScalar performs fast, flexible and accurate simulation of modern processors that implement a derivative of the MIPS-IV architecture [26] and support superscalar, out-of-order execution which is typical for today's high-end processors. The power estimation engine is based on the SimpleScalar architecture, but in addition, it supports detailed cycle-accurate information for all modules, including datapath elements, control and clock distribution network. While being similar to the *Wattch* power simulator [8] in the models used for memory arrays and caches, it has several features that *Wattch* does not support:

- Cycle-accurate power estimation of datapath modules like integer or floating-point ALUs and multipliers.
- Parameterizable clock power modeling as a function of the pipeline depth and number of pipeline registers that need to be clocked.

We would like to point out that the two-level simulation approach is completely general and applicable to any detailed power/

1.A *basic block* is a straight-line piece of code ending at any branch or jump instruction.

performance simulation engine, as long as it is augmented with a hotspot detection mechanism for speed-up purposes.

In what follows, we present details on hotspot detection mechanics, how is sampling done inside a hotspot and power modeling for the core modules.

4.1 Identifying hotspots

As described previously, collections of basic blocks executing very frequently together are called *hotspot*s. It has been shown that most of the execution time of a program is spent in several small critical regions of code, or in several *hotspot*s. These hotspots consist of a number of basic blocks exhibiting *strong temporal* locality. In [25] a scheme for detecting hotspots at run-time has been presented. Our purpose is to use the *hotspot detection mechanics* in a simulation environment so as to speed-up power/performance estimation in a design exploration environment. To the best of our knowledge, this has not been attempted before. As opposed to previous approaches which implement the hotspot detection and monitoring mechanism *in hardware*, we implement them within the simulator itself. The main advantage is that the overhead introduced in simulation time is negligible, as we shall see later.

```
hotspot_detect (BBB, HDC, br) {
 if br is not in BBB {
  if not a conflict
   insert br in BBB;
  else return;
 }
 BBB(br).ExecCtr ++;
 if BBB(br).ExecCtr > BBB_Threshold
  BBB(br).CF = true;
 if BBB.CF(br) == true
  HDC = HDC - D;
 else
  HDC = HDC + I;
 if HDC == 0
  In_Hotspot = true;
 else
  if HDC == Max_HDC and In_Hotspot == true
   In_Hotspot = false;
}
```

Fig.5 The hotspot detection scheme

To keep track of branches, we use a cache-like data structure called the *branch behavior buffer* (*BBB*). Each branch has an entry in the *BBB*, consisting of an execution counter (*ExecCtr*) and a one-bit candidate flag (*CF*). The execution counter is incremented each time the branch (*br*) is taken, and once the counter exceeds the BBB_Threshold value (512, in our case), the branch in question is marked as a *candidate branch* by setting the *CF* bit for that branch. Fig.5 shows the details of the hotspot detection scheme. Specifically, in addition to the *BBB* structure we include the following into our simulation engine:

- A *saturating* counter called the *hotspot detection counter* (*HDC*) keeps track of candidate branches. Initially, the counter is set to a maximum value (Max_HDC); each time a candidate branch is taken, the counter is decremented by a value *D*, and each time a non-candidate branch is taken, it is incremented by a value *I*. When the *HDC* decrements down to zero, we are in a hotspot. For our implementation we chose *D* as 2 and *I* as 1, such that exiting the hotspot is twice as slow as entering it.
- The *BBB* and *HDC* are left running even when execution is inside the hotspot. When the code strays away from the hotspot, non-candidate branches start to execute more frequently; the *HDC* then increments to its upper limit eventually, and we say that we are out of the hotspot.
- The replacement policy for entries in the *BBB* is that if there is a conflict, the old entry is retained and the new one discarded. Entries are *not* replaced; this is needed so that the *BBB* figures reflect the correct execution statistics.
- Every several (e.g., 4096, in our case) cycles, *BBB* entries which are non-candidate entries are flushed. If a hotspot is not yet detected, every 64K cycles, the entire *BBB* is reset. These two mechanisms ensure that the replacement policy we have adopted does not cause stagnation of entries in the table.

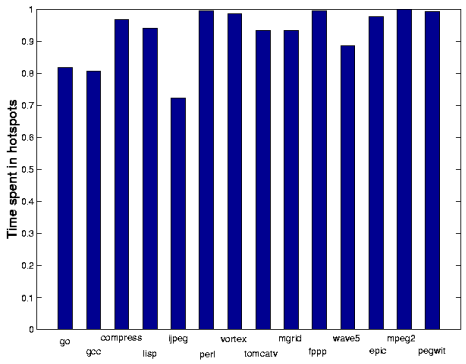

Fig.6 Normalized dynamic execution time spent in hotspots

The amount of time a program spends in hotspots depends on the behavior of the program itself; Fig.6 illustrates this for the benchmarks that we tested. The average fraction of time spent inside detected hotspots is 92%, with the fraction being higher for MediaBench [27] than for Spec95 benchmarks [28].

We should point out that the hotspot detection scheme described here cannot be simply replaced by monitoring branch prediction behavior. While it is true that the entry and exit points are branches that have very predictable behavior, in general, the same cannot be said about internal conditional branches. What the described scheme provides in addition to branch prediction behavior monitoring, is a way of generating collective statistics characterizing basic blocks which are closely coupled one to another. In the following, we describe how metrics of interest can be obtained via *sampling* inside detected hotspots.

4.2 Sampling hotspots

As mentioned before, the main mechanism for achieving speed-up in power /performance simulation is the fast convergence of both *IPC* and *EPC* metrics while code is running inside a hotspot. Our proposed sampling scheme is shown in Fig.7.

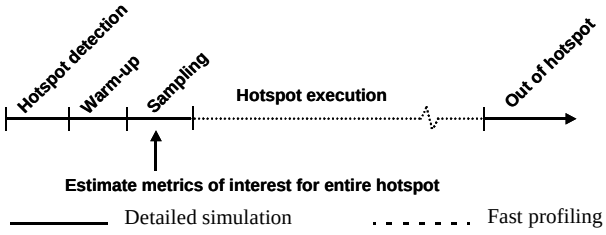

Fig.7 The timeline of events

After a hotspot is detected, no sampling is done for a *warm-up period* needed to bypass any transient regime due to compulsory cache misses, etc. Then, for a number of cycles denoted by the *sampling window size*, metrics of interest (committed instructions, access counts, cache misses, etc.) are monitored and collected in lumped *CPI*, *EPI* or *EDPPI* metrics that characterize the entire hotspot. After the sampling period is over, the simulator is switched to the fast profiling mode and then back to detailed mode when the exit out of the hotspot is detected.

To illustrate this proposed sampling scheme, we have considered three of the SpecInt95 benchmarks that exhibit a different profile as far as parallelism, branch behavior and functionality is concerned: *ijpeg* (an image processing benchmark), *gcc* (GNU C compiler) and *compress* (implementing the Ziv-Lempel compression algorithm). Using the SimpleScalar architectural simulator and our proposed cycle accurate power simulation engine (described next), we show in Fig.8-9 the relative error obtained when different sampling window sizes are used. The results have been obtained by sampling *IPC* and *EPC* values for each detected hotspot for a

sampling window size varying between 32K and 128K cycles. The sampling is started after a warm-up period of 200,000 cycles.

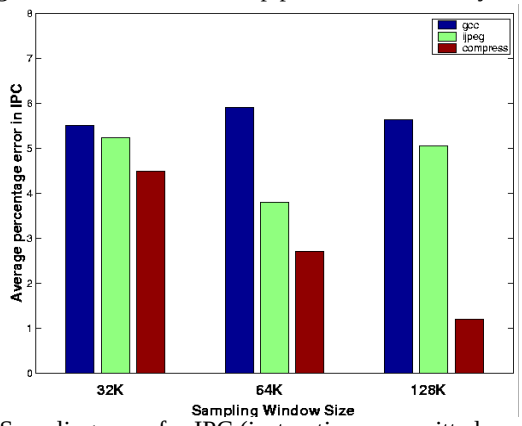

Fig.8 Sampling error for IPC (instructions committed per cycle)

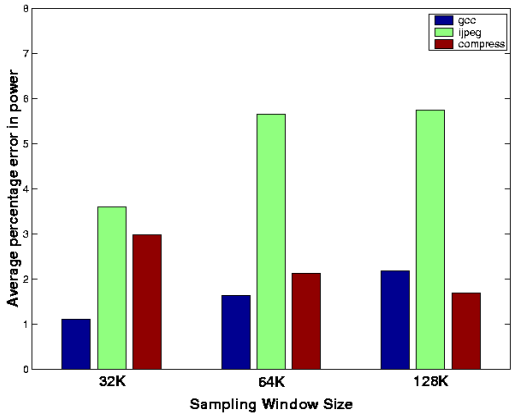

Fig.9 Sampling error for EPC (average power consumption)

The dynamic duration of the hotspots of different benchmarks varies between 1M and 50M cycles. As it can be seen, for both *IPC* (Fig.8) and *EPC* (Fig.9), there is no significant accuracy loss if sampling is done for only 1-5% of the cycles for each hotspot. In fact, the percentage error when compared to the baseline simulator is in all cases less than 6% for *IPC* (4-5% on average, depending on the sampling window size) or for *EPC* (2-4% on average). We note that the error is not necessarily monotonic with increasing sampling window size, due to the different control and data dependencies exhibited by different applications, but is always within 6% when compared to the detailed power/performance simulation engine.

4.3 Microarchitectural power modeling

Our proposed microarchitectural power modeling scheme relies on using activity-driven, parameterizable power models, like *Wattch* [8]. However, unlike *Wattch* which concentrates on accurately modeling the memory arrays using capacitance models very similar to previously proposed *Cacti* tools [22-24], we use in addition cycle-accurate models for datapath modules like integer and floating point ALUs and multipliers. Also, we include *parameterizable models for global clock power* as a function of pipeline depth and configuration.

Specifically, the power models used for the datapath modules are based on input dependent macromodels [29]. The input statistics are gathered by the underlying detailed simulation engine and used, together with technology specific load capacitance values, to obtain power consumption values. Assuming a combination of static and dynamic CMOS implementations, we use a cycle-accurate power macromodeling approach for each of the units of interest [29]:

$$P_{module,\,k} = F_{module}(V_{module,\,k-1}, V_{module,\,k})$$

where $P_{module,k}$ is the power consumption of a given module during cycle k, when input vector $V_{module,\,k-1}$ is followed by $V_{module,\,k}$.

While estimation accuracy is important for all modules inside the core processor, it is recognized that up to 40-45% of the power budget goes into the global clock power [2]. Thus, accurate estimation of the global clock power is particularly important for evaluating power values of different core processor configurations. Specifically, we estimate the global clock power as a function of the die size and number of pipeline registers [30]:

$$P_{clk} = f_{clk} V_{dd}^2 \left(N_r C_r + 1.5(2^h - 1) D C_w + \alpha \sqrt{4^h N_r C_w} \right) \quad (1)$$

where the first term accounts for the register load and second and third account for the global and local wiring capacitance (α is a term which depends on the local routing algorithm used, h is the depth of the H-tree). C_r is the nominal input capacitance seen at each clocked register and C_w is the wire capacitance per unit length, while N_r is the number of pipeline registers, $N_r = \sum_{i=1}^{p} N_i$ for p pipeline stages.

To estimate the die size and number of clocked pipeline registers, we use the microarchitectural configuration as follows:

$$Area = Area_{memory-arrays} + Area_{CAM-arrays} + Area_{Functional-units} + Area_{Regfile} + Area_{clock} \quad (2)$$

where *memory* and *CAM arrays* (Content-Addressable Memory) account for caches, TLBs, branch prediction table, rename logic and instruction window, *functional units* are the integer and floating point units and *clock* refers to the clock distribution tree and clocked pipeline registers. To estimate the size of each module, we rely on the wirelength and module size calculation done in *Cacti* which is at the basis of latency estimation.

The pipeline depth is directly related to the actual latency of different modules and the level of pipelining used for each of them. For example, if a target clock frequency of 200MHz is chosen and the D-cache cycle time is 4500 ps, the D-cache access has to be pipelined in 2 stages to accommodate the given clock frequency. Even without a prescribed clock frequency, balanced pipelining is desired and more pipeline stages may be required to achieve this. To estimate the area of different memory arrays (I-cache, D-cache, TLB, branch prediction table, instruction issue window, etc.) and their latency, we use *Cacti* tools which provide accurate models (within 5-7% error when compared to HSPICE) for estimating load capacitances based on realistic implementations of different memory arrays and CAM structures. In addition, it relies on load calculation based on RC models using wirelength estimation and appropriate scaling among different technologies. Similar models are used for power modeling of array and CAM structures in *Wattch*. For a complete analysis of wirelength, module size or latency, the reader is referred to [22-24,30,31].

5 Core processor design exploration

We describe in this section the core processor design exploration environment. The set of parameters that the design exploration framework considers are *issue width (IW)*, *instruction window size* (*WS*) and *pipeline depth (PD)*.

As pointed out previously, *IW* and *WS* are inter-related since more parallelism (and thus higher *IW*) can be extracted when a larger window size (*WS*) is used. Thus, we have pruned the design space by selecting only (*IW*, *WS*) pairs belonging to a meaningful subset. In addition, the number of pipeline stages is directly related to the end-to-end latency of each module. To achieve higher clock rates, more often designers resort to pipeline stage balancing (by moving logic across stages) or to using deeper pipelining so as to increase clock rate. However, deeper pipelines have the disadvantage of increasing mispredict penalties, in addition to higher global clock power consumption. In addition, in some cases it is not possible to further pipeline or move logic around to achieve balanced pipelines, as is the case of wake-up and select logic. There is a clear *trade-off* between nicely balanced pipelined designs achieving high throughput (and possibly high power consumption) and more power efficient, non-balanced pipelines, with a lot of potential for finely tuning the supply voltage of each stage to the required throughput. Our processor design exploration framework is intended to find the *energyxdelay*

product-optimal core organization via simultaneous microarchitecture configuration, clock frequency and multiple supply voltage selection.

For each selected microarchitecture configuration, the design exploration environment first determines the latency of each module for a given microarchitecture configuration, as described in Section 4. It then tries to balance the pipeline stages by further pipelining accesses to caches, register file or functional units (except wake-up/ select stage since it has to be performed atomically). After the number of pipeline stages is determined, the two-level simulation engine is invoked and power/performance metrics are reported. In case an imbalance is detected in the latency of pipeline stages, the design exploration environment tries to assign lower voltages to faster stages so as to reduce power.

While we do consider assigning different voltages for the purpose of reducing power consumption, we do keep the global clock lines powered at the higher V_{dd}. This will limit the amount of savings that can be achieved since the clock power consumption can be significant (up to 40% [2]).

6 Experimental results

The purpose of our experimental study is threefold:

1. To assess the effectiveness of our proposed scheme for *speeding-up* microarchitecture level simulation.
2. To *explore* the design space based on different microarchitectural configurations in terms of power consumption, performance and energyxdelay product.
3. To assess the *efficiency* of our proposed architectural design exploration framework for multiple supply voltage usage in different microarchitectural settings.

6.1 Efficient microarchitecture simulation

In this section, we provide our results for the two-level simulation environment described in Section 3. The processor configuration considered was a 4-way superscalar with an instruction window size of 32. We have also assumed a register file of size 32, a direct mapped I-cache of size 16K with a block size of 32B and a 4-way set associative D-cache of size 16K with 32B blocks. To report our experimental results, we have used as detailed simulator a modified version of *Wattch* which accounts for data dependent power values and parametrized clock power, as described in Section 3. For the purpose of reporting the results, the simulator was run in the high-performance mode which uses a fairly accurate branch prediction mechanism. For the non-detailed profiling simulation, we have monitored only branch instruction for the purpose of identifying if we are still in or out of a hotspot, as described in detail in Section 3.

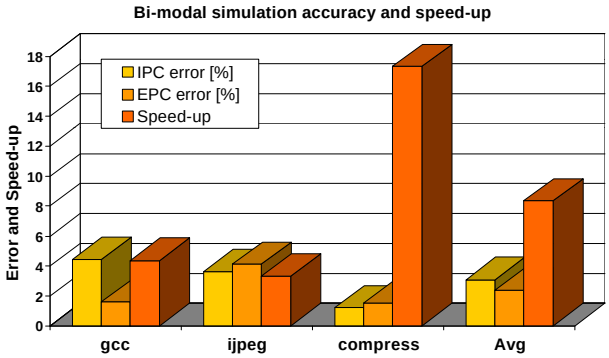

Fig.10 Simulation accuracy and speed-up (128Kcycles sampling)

We present in Fig.10 the accuracy and speed-up of the proposed mixed mode simulation engine, both in terms of power and performance. As it can be seen, since most benchmarks spend most of the execution time in hotspots, using sampling inside hotspots with a sampling window size of 128K cycles provides between 3X and 17X improvement in simulation time (including the overhead due to hotspot detection). In addition, Fig.10 shows the accuracy of this two-level simulation environment. When compared to the original cycle accurate simulator, our proposed two-level simulation is within 3% accurate for *EPC* estimates and 3.5% for *IPC* values. Thus, being both accurate and fast, the two-level approach is an ideal candidate for a design exploration framework, as described next.

6.2 Design exploration

In the following, we present our results on the same subset of the SpecInt95 benchmark suite (*compress*, *ijpeg*, *gcc*). The three benchmarks were chosen so as to exhibit different behavior both in terms of power and performance values when microarchitectural configurations are varied. All other benchmarks fall into one of the presented categories. All metrics have been obtained assuming a 0.35μm technology, with V_{dd} = 2.5V.

As the set of microarchitectural configurations to be explored, we have chosen (*IW*,*WS*) ∈ {(2,16), (4,16), (4,32), (8,32), (8,64), (8,128)}. After the latency analysis step is completed, accesses are further pipelined if needed. As shown in [22-24,30,31], the first candidates for further pipelining are I-cache and D-cache accesses, as well as the execution stage and register file. Doing so, the bottleneck remains the wake-up and select logic which has to be performed atomically and thus may dictate the overall clock frequency. For all configurations considered above, by further pipelining all mentioned stages, we get the following pipeline structure:

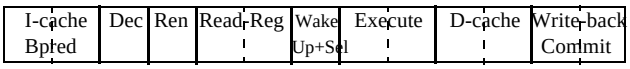

Fig.11 The pipeline structure

However, for the first 4 cases the clock rate is dictated by half of $T_{D\text{-}cache}$, whereas in the last 2 cases ((8,64) and (8,128)) the slowest stage is the wake-up and select. Including the delay of pipeline registers, the performance is given by the number of committed instructions per unit time IPC/T_{cycle}, whereas the energy metrics are given by the average power per cycle (*EPC*), energy per committed instruction ($EPI = EPC * CPI$) and energyxdelay product per committed instruction ($EDPPI = EPI * CPI * T_{cycle}$).

In all cases reported, the best *IPC* is obtained when a wider *IW* and/or a large *WS* is used. *IPC* steadily increases when *IW* is increased, although in some cases (e.g., *gcc*), the dependence on *WS* is more prevalent. However, in most cases, going from a window size of 32 to 64 or 128 brings almost no improvement in terms of performance and it can actually reduce the performance due to a slower clock rate dictated by a very slow wake-up/select logic (as is the case of (8, 128) configuration). On the other hand, the average power consumption (*EPC*) is usually minimized for lower values of *IW* and *WS* but this reduction comes at the price of decreased performance. In fact, the total energy consumed during the execution of a given benchmark may actually increase due to increased idleness of different modules.

To characterize the total energy consumption, the energy per committed instruction (*EPI*) is a more appropriate measure. While in some cases (*compress*, *ijpeg*) *EPI* decreases with higher *IW* and increases with higher values of *WS*, there are cases where *EPI* decreases with increasing *IW* (*gcc*). However, for all 3 benchmarks, the lowest *EPI* configuration is characterized by relatively low values for *IW* and *WS* (4 and 16, respectively). If, however, the highest energy reduction with lowest performance penalty is sought, in all cases but one (*gcc*) the optimal configuration (i.e., lowest energyxdelay product *EDPPI*) is achieved for *IW* = 8 and *WS* = 32. Although the energy is not minimized in these cases, the penalty in performance is less than in other cases with similar energy savings.

Thus, in terms of energy efficiency, the best configuration is not necessarily the one that achieves the highest *IPC* or performance. Depending on the actual power budget, processor designers may choose to go with lower-end configurations, with not too much of a reduction in performance.

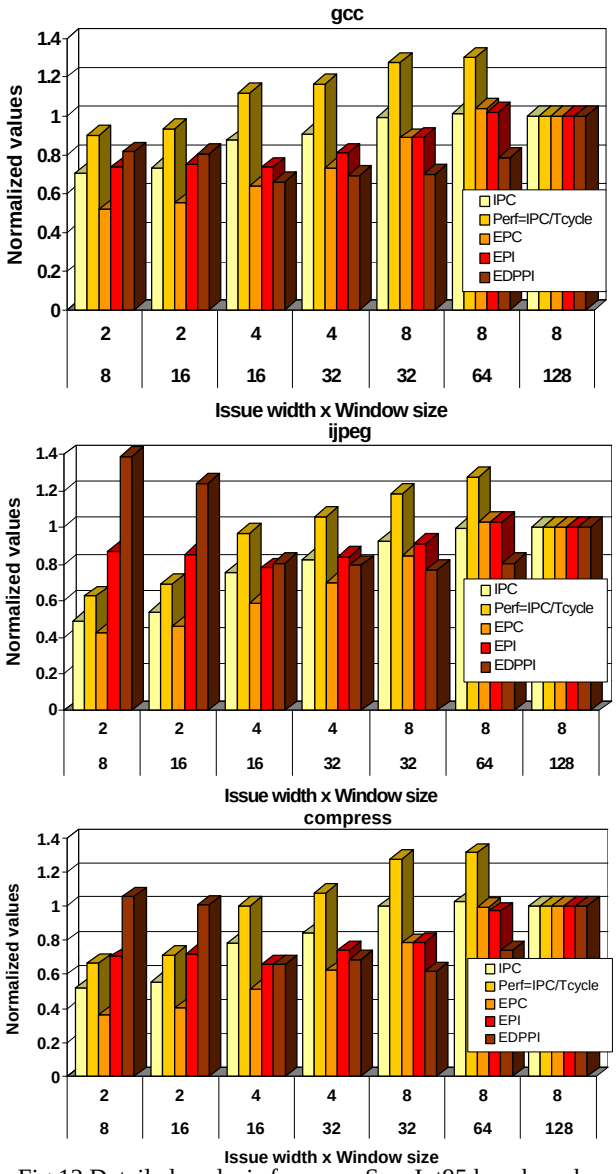

Fig.12 Detailed analysis for some SpecInt95 benchmarks

6.3 Using multiple supply voltages

To this end, we consider the effect of using multiple voltages on a high-end configuration (8-way superscalar, with a 128- entry issue window) assuming that up to three supply voltages are available: V_{dd1} = 2.5V, V_{dd2} = 2V, V_{dd3} = 1.5V. We consider both the case of balanced pipelining (as in Fig.11), as well as the case of an 8-stage pipeline (Fig.13) with latencies per stage computed as in [22-24,31]. For the 3 voltage supply case, we have also included the overhead due to level converters and DC-DC converter [32]. We show in Fig.14-15 the performance (IPC/T_{cycle}) and energyxdelay product per committed instruction *EDPPI* for both cases. .

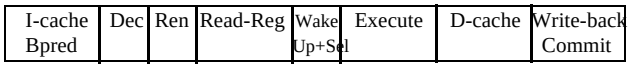

Fig.13 The case of non-balanced pipeline

As expected, balancing stages via finer pipelining increases performance by 40%, and also decreases *EDPPI* by 10-15% at the expense of a larger power per cycle. However, if we assume that three different supply voltages are available (2.5V, 2V, 1.5V), we can run faster stages at lower voltages, and thus lower power, without changing the clock rate of the original pipeline. For the case in Fig.11 (balanced) we have used V_{dd1} for the issue window and global clock, V_{dd2} for D-cache and functional units, and V_{dd3} for the rest of the resources. In the non-balanced case (Fig.13), we have used V_{dd1} for D-cache and functional units, V_{dd2} for issue window and I-cache, as well as V_{dd3} for the rest of modules. As seen in Fig.15, the *EDPPI* decreases by about 30% for non-balanced pipelining and by 23% in the balanced case. We can also see that by using multiple voltages, the gap between the energy efficiency of the 8-stage and 13-stage processor organizations is reduced by more than 50%.

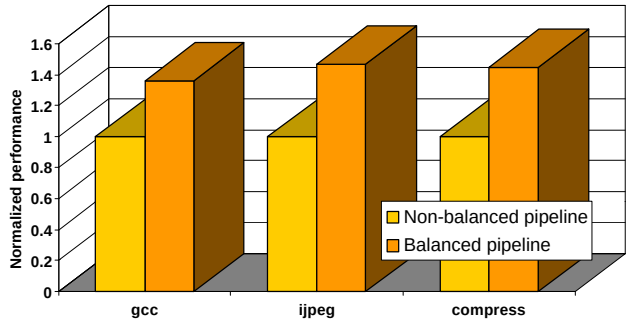

Fig.14 Performance of balanced vs. non-balanced pipelines

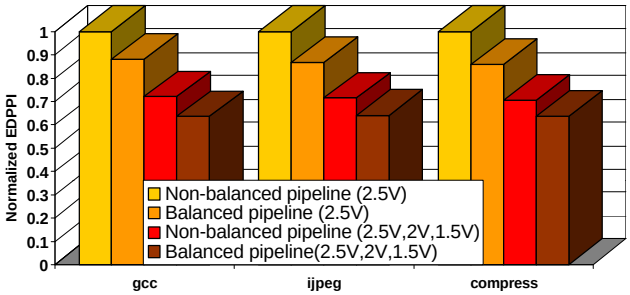

Fig.15 Energyxdelay product using multiple supply voltages

Several conclusions can be drawn from this exploratory analysis:

1. As expected, performance and power efficiency can be severely limited if stage balancing is not possible (see the (8, 128) configuration in Fig.12).
2. Energy metrics are typically minimized for lower-end configurations. In particular, *EPI* (reciprocal of MIPS/Watt) is minimum for a 4-way, 16-entry issue window core, whereas *IPC* is maximum for higher-end configurations. Keeping power consumption under control for these cases may not be possible unless some performance is surrendered, or if the microarchitecture configuration is drastically changed.
3. For very high-end configurations where balancing may no longer be possible, using multiple voltages can improve energy efficiency by about 23-30%, without reducing performance when compared to the original pipeline.

We would like to point out that the exploration of the design space has been performed for all configurations described herein in less than 1.5 hours for all benchmarks considered, whereas using a brute-force approach based on low-level, detailed simulation would have taken at least 12-14 hours.

7 Conclusion and discussion

In this paper, we have presented an efficient design exploration environment for high-end core processors. At the heart of this design exploration framework is a two-level simulation engine that combines detailed simulation for critical portions of the code with fast profiling for the rest. The critical regions of the code are discovered via hotspot detection and are sampled for power/ performance metrics convergence. The proposed simulation methodology is 3-17X faster, while being sufficiently accurate (within 5%) when compared to the fully detailed simulator.

The design exploration environment is able to vary different

energyxdelay product is concerned in a matter of minutes. For very high-end configurations, it was shown that balanced pipelining may not be possible, and thus opportunities for running faster stages at lower voltage exist. In such cases, by using up to 3 voltage levels, the energyxdelay product is reduced by 23-30% when compared to the single voltage implementation.

As a possible future research direction, the paradigm of running slower stages at a lower voltage could also be employed in a run-time environment that is able to adjust the voltage and clock frequency dynamically, on a fine grain, to fit the application needs.

8 Acknowledgments

The first author would like to thank the reviewers for their detailed and constructive feedback.

9 References

[1] J. Mermet and W. Nebel, 'Low Power Design in Deep Submicron Electronics,' Kluwer Academic, Norwell, MA, 1997.

[2] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, F. Baez, 'Reducing Power in High-Performance Microprocessors,' in *Proc. ACM/IEEE Design Automation Conference*, pp.732-737, June 1998.

[3] V. Tiwari, S. Malik, and A. Wolfe, 'Power Analysis of Embedded Software: A First Step Toward Software Power Minimization,' in *IEEE Trans. on VLSI Systems*, vol.2, no.4, pp.437-445, April 1994.

[4] C.L. Su, C.-Y. Tsui, and A.M. Despain, 'Saving Power in the Control Path of Embedded Processors,' in *IEEE Design and Test of Computers*, vol.11, no.4, Dec. 1994.

[5] S.T. Cheng, C.M. Chen, J.W. Huang, 'Low-Power Design for Real-Time Systems,' in *Real-Time Systems*, vol.15, no.2, pp.131-148, Sept. 1998.

[6] M.T.-C. Lee, V. Tiwari, S. Malik and M. Fujita, 'Power Analysis and Minimization Techniques for Embedded DSP Software,' in *IEEE Trans. on VLSI Systems*, vol.5, no.1, pp.123-135, Jan. 1997.

[7] B. Klass, D.E. Thomas, H. Schmit, D.E. Nagle, 'Modeling Inter-Instruction Energy Effects in a Digital Signal Processor,' in *Power-Driven Microarchitecture Workshop*, in conjunction with *Intl. Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[8] D. Brooks, V. Tiwari, and M. Martonosi, 'Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,' in *Proc. Intl. Symposium on Computer Architecture*, Vancouver, BC, Canada, June 2000.

[9] N. Vijaykrishnan, M. Kandemir, M.J. Irwin, H.S. Kim, and W. Ye, 'Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower,' in *Proc. Intl. Symposium on Computer Architecture*, Vancouver, BC, Canada, June 2000.

[10] T. Simunic, L. Benini and G. De Micheli, 'Cycle-accurate simulation of energy consumption in embedded systems,' in *Proc. ACM/IEEE Design Automation Conference*, New Orleans, June 1999.

[11] S. Manne, A. Klauser, and D. Grunwald, 'Pipeline Gating: Speculation Control for Energy Reduction,' in *Proc. Intl. Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[12] T.M. Conte, K.N. Menezes, S.W. Sathaye, and M.C. Toburen, 'System-Level Power Consumption Modeling and Trade-off Analysis Techniques for Superscalar Processor Design,' in *IEEE Transactions on VLSI Systems.*

[13] D. Albonesi, 'Selective Cache Ways: On-Demand Cache Resource Allocation,' in *Proc. Intl. Symposium on Microarchitecture (MICRO-32)*, Haifa, Israel, pp.248-259, Nov. 1999.

[14] J. Kin, M. Gupta, and W. Mangione-Smith, 'The Filter Cache: An Energy Efficient Memory Structure,' in *IEEE Micro*, Dec.1997.

[15] H. Lekatsas, J. Henkel, and W. Wolf, 'Code Compression for Low Power Embedded System Design,' in *Proc. ACM/IEEE Design Automation Conference*, Los Angeles, CA, June 2000.

[16] V. Zyuban and P. Kogge, 'Optimization of High-Performance Superscalar Architectures for Energy Efficiency,' in *Proc. ACM Intl. Symposium on Low Power Electronics and Design*, Portofino, Italy, July 2000.

[17] J. Kin et al., 'Power Efficient Media Processors: Design Space Exploration,' in *Proc. ACM/IEEE Design Automation Conference*, New Orleans, LA, June 1999.

[18] W.-T. Shiue and C. Chakrabarti, 'Memory Exploration for Low Power, Embedded Systems,' in *Proc. ACM/IEEE Design Automation Conference*, pp.140-145, New Orleans, LA, June 1999.

[19] I. Hong et al., 'Power Optimization of Variable Voltage Core-Based Systems,' in *Proc. ACM/IEEE Design Automation Conference*, San Francisco, CA, June 1998.

[20] G. Qu et al., 'Energy Minimization of System Pipelines Using Multiple Voltages,' in *Proc. IEEE Intl. Symposium on Circuits and Systems*, June 1999.

[21] D. Burger, T.M. Austin, 'The SimpleScalar Tool Set, Version 2.0,' *CSD Technical Report #1342*, University of Wisconsin-Madison, June 1997.

[22] S. Palacharla, N.P. Jouppi, and J.E. Smith, 'Quantifying the Complexity of Superscalar Processors,' CS-TR-1996-1328, Univ. of Wisconsin, Nov. 1996.

[23] K.I. Farkas, N.P. Jouppi, and P. Chow, 'Register File Design Considerations in Dynamically Scheduled Processors,' WRL Research Report 95/10, Digital Equipment Corp., Nov. 1995.

[24] S.J.E. Wilton and N.P. Jouppi, 'An Enhanced Access and Cycle Time Model for On-Chip Caches,' WRL Research Report 93/5, Digital Equipment Corp., July 1994.

[25] M.C. Merten, A.R. Trick, C.N. George, J.C. Gyllenhaal, and W.-M. Hwu, 'A hardware-driven profiling scheme for identifying program hot spots to support runtime optimization,' in *Proc. Intl. Symposium on Computer Architecture*, June 1999.

[26] C. Price, 'MIPS IV Instruction Set, revision 3.1.,' MIPS Technologies, Inc., Mountain View, CA, Jan. 1995.

[27] C. Lee, M. Potkonjak, and W. Mangione-Smith, 'MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems,' in *Proc. Intl. Symposium on Microarchitecture*, Dec. 1997.

[28] *Spec'95 Benchmark Suite*, `http://www.spec.org`

[29] C.-Y. Tsui, K.-K. Chan, Q. Wu, C.-S. Ding, and M. Pedram, 'A Power Estimation Framework for Designing Low Power Portable Video Applications,' in *Proc. ACM/IEEE Design Automation Conference*, San Diego, June 1997.

[30] C. Svensson and D. Liu, 'Low Power Circuit Techniques,' in *Low Power Design Methodologies* (Eds. J.M. Rabaey and M. Pedram), pp.37-64, Kluwer Academic, Norwell, MA, 1996.

[31] *Cacti 2.0 Technical Report*, `http://www.research.compaq.com/wrl/people/jouppi/cacti2.pdf`

[32] M. C. Johnson and K. Roy, 'Datapath Scheduling with Multiple Supply Voltages and Level Converters,' in *ACM Trans. on Design Automation of Electronic Systems*, Vol.2, No.3, July1997, pp. 227–248.

[33] J. Huang, D.J. Lilja, 'Extending Value Reuse to Basic Blocks with Compiler Support,' in *IEEE Trans. on Computers*, vol.49, No.4, Apr. 2000.

[34] R. Marculescu, D. Marculescu, and M. Pedram, 'Sequence Compaction for Power Estimation: Theory and Practice', in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.18, No.7, July 1999.

[35] C.-S. Ding, Q. Wu, C.-T. Hsieh, M. Pedram, 'Stratified random sampling for power estimation,' in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.17, No.6, June 1998.

[36] P.K Dubey and R. Nair, 'Profile-driven Generation of Trace Samples,' in *Proc. IEEE Intl. Conf. on Computer Design: VLSI in Computers and Processors*, Oct. 1996.

[37] A.-T. Nguyen, P. Bose, K. Ekanadham, A. Nanda, M. Michael, 'Accuracy and Speed-up of Parallel Trace-Driven Architectural Simulation,' in *Proc. IEEE Intl. Symposium on Parallel Processing*, 1997.

[38] V.S. Iyengar, P. Bose, and L. Trevillyan, 'Representative Traces for for Processor Models with Infinite Cache,' in *Proc. ACM Intl. Symposium on High-Performance Computer Architecture*, Feb. 1996.

[39] C.-T. Hsieh, M. Pedram, 'Microprocessor Power Estimation Using Profile-Driven Program Synthesis,' in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.17, No.11, Nov. 1998.