Appeared in Koolchips Workshop, during MICRO-33, Monterey, California, December 2001

Adapting Processor Supply Voltage to Instruction-Level Parallelism

Bruce R. Childers, Hongliang Tang, Rami Melhem
Department of Computer Science
University of Pittsburgh, Pittsburgh, PA 15260
{childers, tanghl, melhem}@cs.pitt.edu

Abstract

Low power consumption is one of the most important design constraints for modern computer systems. One promising technique to lowering power consumption of microprocessors is to reduce supply voltage and clock speed. By running at lower voltages and clock speeds, a quadratic savings in power can be achieved. This paper describes a technique for adjusting supply voltage and frequency at run-time to save energy. Our technique monitors a program's instruction-level parallelism (ILP) and adjusts processor voltage and speed in response to the amount of observed ILP. The technique lets the user specify goal performance, which the hardware maintains while running at the lowest energy settings. In comparison to a processor running at a fixed voltage and speed, our approach improves energy consumption by an average of 47%.

1. Introduction

In recent years, power consumption has become a major concern for system designers. For battery-operated systems, the focus on reduced power comes from a desire for greater device capability and longer missions with smaller form factors, less weight, and reduced cost. In these systems, the size of the battery is one of the most important concerns. However, a fast processor that is capable of handling a variety of tasks has considerable power demands, and devices using such processors need large and expensive batteries. For servers, reducing power has the effect of increasing server density and improving reliability, both important for large data centers. For example, an Internet service provider with 8,000 servers needs 2 megawatts of power [2]. It is clear that both personal electronics and servers can benefit greatly from processors that consume less energy.

In this paper, we study how changing processor supply voltage can conserve energy. In particular, we explore whether changing supply voltage in response to changes in instruction-level parallelism (ILP) can improve the energy consumption of out-of-order superscalar processors. Our initial scheme relies on the user (or operating system) expressing a preference about a desired level of performance. This preference represents a quality of service (QoS) metric, where the user expresses that he is willing to execute a program at a particular rate. For example, an e-mail client does not need a 1 GHz processor to run effectively. Our goal is to give the user the ability to tell the system that it may operate at lower performance levels to save energy.

Because modern systems run varying workloads, there may be applications which require high performance at any cost. Our approach lets the user express QoS concerns to the underlying hardware, so the hardware can adjust performance to meet QoS goals while running at the lowest energy settings. This gives the user direct control over how fast to run a program and to make trade-offs with performance and energy. Such an ability benefits both battery-powered systems and servers. For personal electronics, the battery life will be increased, and for servers, reliability will be increased (by reducing heat dissipation). We have implemented a technique for adjusting supply voltage in a microarchitecture simulator. We found that dynamically changing supply voltage improves processor energy consumption by an average of 47%.

Supported under DARPA contract No. F336–15–00–C–1736.

2. Energy Consumption

The amount of power consumed by a portable system is useful for determining the maximum current that a battery must supply. It is also important for servers in determining the type of cooling features needed to handle a system's expected total heat dissipation. Along with power consumption, an electronic system's energy should be considered because it measures power consumption over time: $energy = power \times time$.

We need to be careful that implementing architectural features which reduce power consumption do not increase execution latency so much that there is no net decrease in energy consumption. E.g., a 50% reduction in power consumption that comes at the cost of a 200% increase in execution latency results in no net gain in battery life. In CMOS circuits, the power consumption due to dynamic switching dominates the power lost to static leakage [4]. For CMOS, the dynamic portion of power consumption, called P_{avg}, is approximated by:

$$P_{avg} = \alpha_T \cdot f_{CLK} \cdot C_{load} \cdot V^2_{DD},$$

where α_T is the activity factor (amount of switching), f_{CLK} is the clock frequency, C_{load} is the capacitance load, and V_{dd} is the supply voltage. Processor power can be decreased by tackling any one of these terms. Indeed, several studies have looked at reducing the activity factor [9], especially for memory [1,5,8]. Reducing the clock speed is also an important way to save energy. However, lowering clock speed without changing supply voltage will not necessarily result in any energy savings. When clock speed and supply voltage are reduced together, there is a quadratic reduction in power consumption. In this paper, we take advantage of this property by dynamically scaling V_{dd} in response to ILP and user preferences for performance.

3. Adaptive Voltage Scaling

Adaptive voltage scaling adjusts supply voltage (and clock speed) in response to some dynamic measure. In this paper, we focus on adjusting supply voltage in response to changes in ILP over time. The essential idea is that an application typically does not need all of the performance offered by a very fast wide-issue processor. Yet modern electronic systems run workloads whose performance demands typically vary and can not be predicted in advance. In this way, systems are overspecified with high performance architectures to handle those applications which require fast execution rates. In terms of energy, overspecifying processor performance comes at a large cost. By running at fast clock rates and correspondingly high supply voltages, an out-of-order architecture wastes energy, especially given the energy costs of architectural features such as branch prediction, large instruction windows, and multiple functional units. For applications that do not need high performance, we want to run slower with smaller voltages to save energy.

Changing processor speed has been studied in the context of real-time systems, where the goal is to run the processor at the slowest speed that ensures that a task's deadline is met [6,7,11]. The control of processor speed is usually made by the operating system since the OS has knowledge of tasks and their deadlines. However, even within a single application, we can adjust speed and supply voltage to save energy. The amount of ILP exhibited by an application changes over time depending on the impact of branch predictions, cache misses, resource contention, etc. To maintain goal performance we want to adjust supply voltage in response to the ILP of an application. For example, if there are many branch mispredictions over a period of time, then the amount of ILP may be reduced. In this case, the processor should run fast to make up for limited ILP. In other code fragments there may be much more ILP, so the processor can run slower to achieve its goal performance.

Adapting speed in response to ILP to maintain goal performance has two benefits. The first benefit is we can smooth application performance by changing processor speed dynamically. Smoothing performance can help applications whose performance varies dramatically over time. For these applications, such as video playback, peaks and valleys in performance can cause "jerkiness" in their QoS. Adapting processor speed on-the-fly helps to smooth out these peaks and valleys.

The second advantage to voltage adaptation is energy savings. By trading parallelism for clock speed, we can save energy quadratically. One way to achieve similar savings without adaptation is to use a lower performance processor with a lower fixed voltage. However, for a system that has a mixed workload, its speed is typically determined by the application that needs the fastest performance. Another approach is to use a processor that allows changing speed on application boundaries. The trouble with this technique is the performance demands of an application may not be known in advance. For some applications, it may be possible to use profiling techniques to get an estimate of performance and change the processor speed in accordance with that estimate before running the program. Even in this case, it's likely that the processor speed is overspecified. Indeed, the performance of an application can be dependent on the input data set, which makes profiling more difficult. Furthermore, in response to a user's willingness to accept lower levels of performance, we would like to provide a mechanism for the user to say that they want some desired QoS. Dynamically changing supply voltage achieves this by adapting on-the-fly to the user's performance goals and to the parallelism exhibited by an application.

4. Voltage Adjustment

Some current processors support dynamic voltage scaling, including Transmeta's TM 5400, Intel's XScale (a StrongARM derivative), and AMD's K6-IIIE+. AMD's processor demonstrates the typical way voltage is changed dynamically. The K6-IIIE+ uses a technology called "PowerNow!" that defines an architectural interface for changing voltage and clock speed (see `www.amd.com`). This technology permits dynamically changing processor speed and voltage to a number of different settings. Current systems however suffer from a high latency penalty for changing supply voltage. PowerNow! uses an external DC-DC regulator to change voltage. Essentially, to change voltage in the K6-IIIE+, a program writes a "voltage value" to a control register and asserts a bit in the processor control register to signal a "change voltage" request. The processor drains the instruction pipeline and sends the value of the control register to an external DC-DC regulator to change supply voltage. During the period when the voltage is changing and stabilizing, the processor "sleeps". For PowerNow!, this sleep period is 75–150 μs, depending on the regulator used. With such high latencies it is difficult to change voltage frequently. In the past, this has been one reason for letting the OS manage processor speed.

Although current mainstream architectures use an external regulator to change voltage, we expect in the near future that processors will have multiple input supply voltages from which they can pick and choose. Multiple voltages already exist on many CPUs: I/O typically uses one voltage (e.g., 3.3 volts), while the processor core uses another (e.g., 1.9 volts). There has also been work in circuit-level voltage scaling, where voltage is matched to the latency of a circuit path [10]. This scaling uses several voltages on chip to run different circuit paths at varying speeds. By bringing multiple voltages on chip, we expect that a processor will be able to quickly change its voltage level.

In this initial study, we assume an architectural interface for changing voltage that is similar to AMD's interface. In our system, the processor measures ILP over some specified interval of time in terms of a MIPS rate. If the MIPS rate changes appreciably either up or down, the processor adjusts its supply voltage accordingly. Before adjusting the supply voltage, the instruction pipeline is drained. After draining the pipeline, the voltage is set to the desired level.

For the experiments in this paper, we measured ILP over a 2 µs interval to decide whether to change speed. Our processor model measures ILP by setting a watchdog timer that interrupts the processor's execution every 2 µs. The watchdog interrupt handler is shown in Table 1. When the watchdog timer expires, the processor calculates the observed ILP over the previous interval as a MIPS rate. Using this MIPS rate, a new frequency (and, hence, voltage level) is calculated. We calculate the new frequency using:

$$f_{new} = f_{old} \times \frac{MIPS_{goal}}{MIPS_{observed}}$$

If f_{new} is high or low enough, the processor selects a new voltage level that achieves the desired frequency. For our model, we assume discrete voltage levels from which the processor can choose. There are 16 levels ranging from 1.65 V at 700 MHz to 1.1 V at 200 MHz in steps of 33 MHz. These levels correspond to those published by Transmeta at the launch of their TM 5400 processor. In this preliminary study, we do not consider the overhead of computing f_{new} and changing the voltage.

There is an important consideration when changing processor speed. Because the speed of external memory is not changed when the processor's speed is changed, the relative difference in latency between the processor and the memory changes. For example, if clock frequency goes from 250 MHz to 500 MHz, the effective memory latency (in CPU clock cycles) doubles. This change in relative memory latency may impact processor design. We want the architecture to scale its central reservation window and instruction fetch logic with a change in processor speed. When running at high speeds, more hardware resources are needed to cover the large CPU-memory latency gap. When running at low speeds, the architecture does not need as large and aggressive structures to mask the CPU-memory latency gap. Indeed, it is particularly desirable to scale the instruction fetch and issue logic since fetch and issue use considerable energy.

```
when watchdog_interval expires {
  observed ← committed_instrs / 2 µs;
  new_freq ← ceil(freq * goal / observed);
  if (new_freq < freq - 33 MHz ||
      new_freq > freq + 33 MHz) {
    stop_instruction_fetch();
    wait_for_pipeline_to_drain();
    level ← get_discrete_setting(new_freq);
    voltage ← voltage_table[level];
    freq ← frequency_table[level];
    resume_instruction_fetch(); }
  committed_instrs ← 0;
  watchdog_interval ← 2 µs; }
```

Table 1: Watchdog Timer Handler

In this study, we do not adjust the hardware when changing processor speed. However, we do model the difference in latency. In our simulations, we scale the number of processor cycles required to make an access to main memory while also scaling processor voltage. We model the difference in latency for all external memory accesses, including TLB misses.

5. Experimental Results

To study the impact of run-time adjustment of supply voltage on processor energy, we added dynamic voltage scaling to the Wattch processor simulator [3]. For this study, we used benchmarks from SPEC95 and MediaBench. We simulated a four-way out-of-order superscalar processor with 64K L1 I and D caches, a 256K L2 unified cache, and a 16 entry instruction window.

Using our simulator, we evaluated whether scaling voltage in response to ILP improves energy consumption. Figure 1 shows energy improvement for the benchmarks. To determine energy improvement, we compared the energy consumption of a processor with voltage scaling (called a "VS processor") against one that ran at a fixed clock speed and voltage. In Figure 1, each set of bars is labelled with goal performance and the clock speed of the baseline. For example, "800 MIPS/700 MHz" means the VS processor tries to achieve 800 MIPS and the baseline runs at 700 MHz, 1.65 V. For "700 MIPS/600 MHz", the baseline runs at 600 MHz, 1.6 V, and for "600 MIPS/500 MHz", the baseline runs at 500 MHz, 1.5 V. In all cases, the VS processor runs at 200–700 MHz and 1.1–1.65 V.

For most benchmarks, dynamically scaling voltage reduces energy consumption. From Figure 1, energy

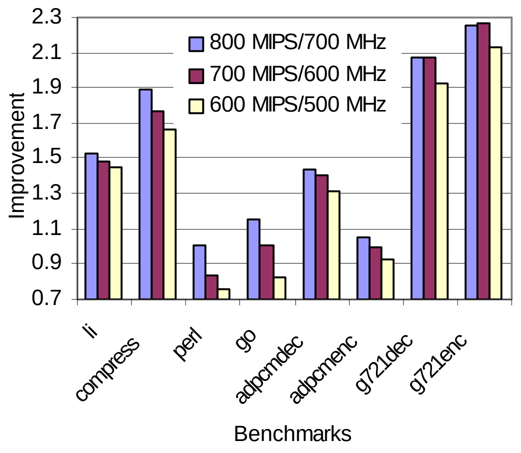

Figure 1: Energy Improvement

improves from 0.76 to 2.27 times, with an average of 1.47. The benchmarks that do the best have enough ILP to trade speed for parallelism. For the benchmarks that have an energy degradation, there is less ILP. For *go* and *adpcmenc*, goal performance of 600 or 700 MIPS causes the VS processor to run at clock rates higher than the fixed frequency baselines. These benchmarks spend 57–61% of their cycles at a clock rate that is higher than the baselines' clock rate. Another reason for possible energy degradation is we use performance over a previous time interval to predict performance of the next interval. For applications that have varying patterns of ILP, using an earlier interval to predict a future one is likely to be ineffective.

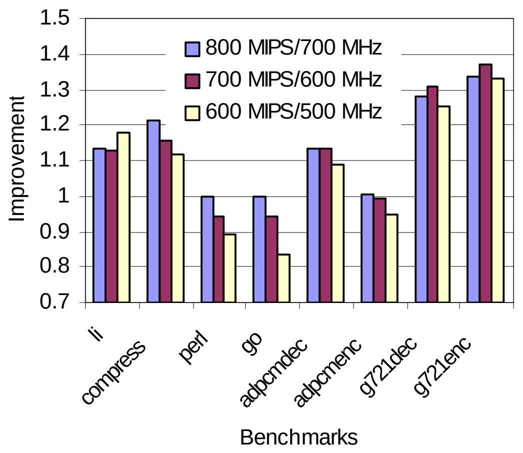

Figure 2: *Energy*×*Delay* Improvement

The metric *energy*×*delay (E•D)* gives an idea of how both execution latency and energy are affected by dynamic voltage scaling. Figure 2 shows that the improvement in *E•D* varies from 0.83–1.37 (average is 1.11). From Figures 1 and 2, the improvement in energy did not adversely hurt execution latency for *li*, *compress*, *adpcmdec*, *g721dec*, and *g721enc*. These benchmarks have enough parallelism that goal performance can be achieved while running at slow clock rates. For *perl*, *go*, and *adpcmenc*, the low *E•D* is due to a lack of an energy improvement. Indeed, these benchmarks have similar or better execution latencies than the fixed frequency baselines. From Figures 1 and 2, we conclude that dynamically adjusting processor supply voltage and clock rate in response to ILP is an effective way to reduce processor energy. This technique is particularly well suited to applications that have much ILP.

6. Summary

This paper describes a technique for dynamically adjusting processor supply voltage and clock speed in response to changes in ILP. By changing supply voltage and clock speed, an aggressive superscalar processor can maintain a goal level of performance while consuming less energy. Our technique reduces energy by up to 47% for a quad-issue superscalar processor.

References

[1] Bahar R. I., Albera G. and Manne S., "Power and performance trade-offs using various Caching Strategies", *Int'l. Symp. on Low-Power Electronics and Design*, 1998.

[2] Benini L. and De Micheli G., "System-level power optimization: Techniques and tools", *ACM Trans. on Design Automation of Electronic Systems*, 2000.

[3] Brooks D., Tiwari V., and Martonosi M., "Wattch: A framework for architectural-level power analysis and power optimization", *Int'l. Symp. on Computer Architecture*, 2000.

[4] Chandrakasan A. and Brodersen R., *Low Power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[5] Childers B. and Nakra T., "Reordering memory bus transactions for reduced power consumption", *ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, 2000.

[6] Hong I., Potkonjak M., and Srivastava M.B., "On-line scheduling of hard real-time tasks on variable voltage processors", *Int'l. Conf. on Computer-Aided Design*, 1998.

[7] Hong I., Qu G., Potkonjak M., and Srivastava M., "Synthesis techniques for low-power hard real-time systems on variable voltage processors", *Proc. of the 19th IEEE Real-Time Systems Symposium*, 1998.

[8] Kin J., Gupta M., and Mangione-Smith W. H., "The filter cache: A energy efficient memory structure", *IEEE/ACM 30th Int'l. Symp. on Microarchitecture*, 1997.

[9] Manne S., Klauser A., and Grunwald D., "Pipeline gating: Speculation control for energy reduction", *Int'l. Symp. on Computer Architecture*, 1998.

[10] Marculescu D., "Power efficient processors using multiple supply voltages", *Workshop on Compilers and Operating Systems for Low Power*, during *PACT'2000*, 2000.

[11] Daniel Mossé, Hakan Aydin, Bruce Childers, and Rami Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications", *Workshop on Compilers and Operating Systems for Low Power*, during *PACT'2000*, 2000.