Instruction Scheduling for Low Power Dissipation in High Performance Microprocessors

Mark C. Toburen   Thomas M. Conte
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina 27695-7911
{mctobure, conte}@eos.ncsu.edu

Matt Reilly
Digital Equipment Corporation
Shrewsbury, Massachusetts
reilly@rock.enet.dec.com

Abstract

Power dissipation is rapidly becoming a major design concern for companies in the high-end microprocessor market. The problem now is that designers are reaching the limits of circuit and mechanical techniques for reducing power dissipation. Hence, we must turn our attention to architectural approaches to solving this problem. In this work we propose a method of instruction scheduling which limits the number of instructions which can be scheduled in a given cycle based on some predefined per cycle energy dissipation threshold. Through the use of a machine description [8], [9] we are able to define a specific processor architecture and along with that an energy dissipation value associated with each functional unit defined therein. Through careful inspection, we can define the cycle threshold such that the maximal amount of energy dissipation can be saved for a given program while incurring little to no performance impact.

1   Introduction

Power dissipation is becoming a vital design issue in today's high-end microprocessor industry. Two examples of high-end processors that suffer from high levels of power dissipation are the DEC 21164a and 21264. The 21164a runs at a clock speed of 600 MHz while dissipating 45.5 Watts of power. The 21264 suffers even worse with internal clock speeds which can reach 666 MHz while dissipating 72 Watts. The problem now is that as power dissipation continues to rise, we are rapidly approaching a point where we will be forced to use cooling techniques which are not suitable for today's personal computers, workstations, and low-end to mid-range servers such as liquid immersion or jet impingement. Circuit designers and thermal engineers have produced some excellent techniques for keeping power dissipation to a minimum in recent years. Clock gating, power supply reduction, smaller process technology, and state of the art packaging are all examples of the approaches that have been used to date to eliminate the power dissipation problem. However, these approaches are reaching their limitations as the demand for processors with higher clock speeds and denser transistor counts continues to rise.

In this work we propose an architectural/compiler approach towards solving this problem. One way we can reduce peak power dissipation is by preventing the occurrence of current spikes during program execution. If a region of code has a heavy profile weight and contains one or more instructions which require significantly more energy than others, then the execution of this region will lead to repeated current spikes in the processor which results in increased power dissipation. Our goal is to prevent this from occurring by limiting the amount of energy that can be dissipated in any given cycle. Because of schedule slack, this often results in little or no performance impact. In our scheduling model, we schedule as many instructions

as possible in a given cycle until the cycle threshold is violated. Once that point is reached, we move on to the next cycle and resume scheduling with the instruction that caused the violation in the previous cycle.

1.1   Previous Work

There have been previous attempts at using scheduling techniques to reduce total power consumption. Su, Tsui, and Despain proposed a technique which combined Gray code addressing and a method called *cold scheduling* to reduce the amount of switching activity in the control path of high performance processors, [1]. Used in conjunction with the traditional list scheduling algorithm, cold scheduling schedules instructions in the ready list based on highest priority. Priority of an instruction is determined by the power cost when the instruction in question is scheduled following the last instruction. The power cost is taken from a power cost table which holds power entries, S(I,J), corresponding to the switching activity caused by the execution of instruction I followed by instruction J. Instructions in the ready list with lower power costs have higher priority. After each instruction is scheduled, the power cost of the remaining instructions in the ready list has to be recalculated before scheduling the next instruction. The drawbacks to the cold scheduling approach are obvious. First, a large table is required to hold power costs for all possible instruction combinations. For a high performance processor with a complex instruction set, this table can be extremely large. Second, this table must be accessed for all instructions in the ready list after each new instruction is scheduled. This will make the scheduling process itself slower. However, Su, Tsui, and Despain show that the combination of Gray code addressing and cold scheduling results in a 20-30% reduction in switching activity for the control path.

Another scheduling technique for reducing power consumption was presented by Tiwari, Malik, and Wolfe, [2], [3]. The goal in these works is to schedule code such that instructions are more judiciously chosen as opposed to instruction reordering. In this approach, actual current measurements were taken on general-purpose processors and DSP processors. Current was measured for each instruction and a power table built for single instruction values as well as values for common paired instructions. Then based on these measurements, test programs were rescheduled to use instructions which result in less power consumption. This selection is based on a number of issues such as register accesses as opposed to memory accesses and lower latency instructions. Tiwari, et al., also take into consideration what they term *circuit-state overhead* which is the switching activity between a pair of specific instructions. In [2] and [3], they argue that for the processors tested that circuit-state overhead is insignificant. However, a detailed analysis of another DSP processor [4], found that circuit-state overhead was much more significant in determining the energy consumption of a pair of instructions. Through this approach of physical measurement and code rescheduling, energy savings up to 40% were achieved on the benchmarks used. Again the problems with this approach are glaring. The process of hand measuring current for all instructions and instruction pairs, while extremely valuable, is extremely time consuming especially in light of the enormous instruction sets used in some of today's high-performance processors. Also, like Su, et al., there is a large table needed to store all power values. This can lead to costly accesses during the scheduling process.

The scheduling approach proposed here is focused on reducing power dissipation as opposed to power consumption. One advantage that our approach provides is that we can explicitly control the amount of power dissipation allowed for any given schedule. The two prior works simply limit power consumption as best they can. In our approach we determine how much power is dissipated which gives us tremendous flexibility in terms of being able to control dissipation for different architectures.

The remainder of this work presents our method of low-power scheduling. Section 2 presents the algorithm itself along with a discussion of the machine description mechanism. Section 3 presents the results of our preliminary investigation into this approach, and in Section 4 we conclude the paper and present plans for future work. All studies presented in this paper were performed using the experimental LEGO compiler designed by the TINKER Research Group at North Carolina State University.

2 Low-Power Scheduling

The goal of traditional scheduling algorithms is to improve performance in terms of execution time. This can be done in a number of ways. Such modern approaches as superblock scheduling [5], hyperblock scheduling [6], and treegion scheduling [7] focus mainly on increasing performance through increasing the amount of instruction-level parallelism in program code. In order to schedule for reduced power dissipation, we are forced to sacrifice some of the performance gains provided by these scheduling algorithms in order to obtain the desired reduction in power dissipation. However, the scheduling approach presented here has shown that significant energy savings can be obtained with minimal reduction in program performance.

In this section the method behind out approach will be presented. First we will discuss the machine description mechanism and how energy values are defined therein. Following will be a discussion of the scheduling algorithm itself.

2.1 MDES Machine Description

We use the MDES machine description language developed at the University of Illinois [8], [9] as the basis for defining the architecture for which we are scheduling. We have built the MDES environment into the LEGO compiler which allows us great flexibility in defining new, experimental architectures. In addition it provides a nice mechanism for defining new machine-specific parameters such as energy dissipation values. In the MDES description, we are able to define hardware resources such as registers, register files, different types of functional units, etc. In addition we can define each operation and the functional units that it can be executed on. It is in this description that we define the energy dissipation values associated with each of the machine's functional units. Once these numbers are defined, the MDES environment builds a data structure that contains the energy information. Then, for each instruction, the scheduler queries the MDES to determine which FU the instruction is to be scheduled on. Once the scheduler knows which FU to scheduler on, it queries the MDES again to get the energy dissipation value associated with the specified FU.

For this particular study the energy values used in the MDES descriptions were obtained from actual power simulations run on different function unit types designed at Digital Equipment Corporation. The numbers provided were abstracted a bit in order not to reveal proprietary information, but are accurate enough to provide reliable results.

2.2 Scheduling Algorithm

The scheduling algorithm presented here is based on the traditional list scheduling algorithm. Once the DAG has been built for a specific region, the list scheduler builds the ready list and begins scheduling instructions based on dependence height. Currently we are only scheduling instructions at the basic-block level. Once an instruction has been cleared to be scheduled and assigned to the proper FU, the FU's energy dissipation value is queried by the list scheduler from the MDES. The list scheduler then adds the value provided to the energy total for the current cycle. If the total exceeds the threshold defined, then the scheduler quits scheduling for the current cycle and begins scheduling for the next cycle with the instruction that caused the violation in the previous one. If the list scheduler does not detect a violation, it proceeds normally.

The results presented in the following section show that this is a powerful technique for reducing power dissipation. We are currently in the process of investigating further enhancements to the current implementation.

3 Experimental Results

The studies performed so far with our scheduler have been run entirely on basic-block code on an 8-issue VLIW architecture defined as Tinker-8 which contains three integer ALU units, 2 general purpose floating-point units, two load/store units, and one branch unit. All results given in this section are for the SPECint95 suite of benchmarks. So far, we have found that careful determination of the per cycle energy threshold can result in significant savings in terms of overall energy violations and total energy saved over a given benchmark. The threshold can take on any value equal or greater to the largest energy value associated with any defined FU. At the low end of the threshold spectrum, we see the optimum amount of energy savings. However, choosing extremely low power thresholds will result in a large impact on program performance in terms of total cycle count. In contrast, selecting extremely high threshold values can result in little to no energy savings. The ideal is to find the point at which energy savings are significant while maintaining program performance. Figures 1 and 2 show results for two test cases in which we chose energy thresholds of 10 and 12 nJ per cycle. Figure 1 shows the total amount of energy violations saved by using the proposed low-power scheduling technique. In general, as the threshold decreases, the number of total violations saved increases. However, the cost is increased execution time. We found that by increasing the threshold from 10nJ to 12, that we could significantly reduce the performance impact while maintaining significant overall energy savings. The overall improvement in execution time ranged from 10.2% faster for 132.ijpeg to 1.3% faster for 130.li.

Figure 1

Total Energy Savings with Low-Power Scheduling
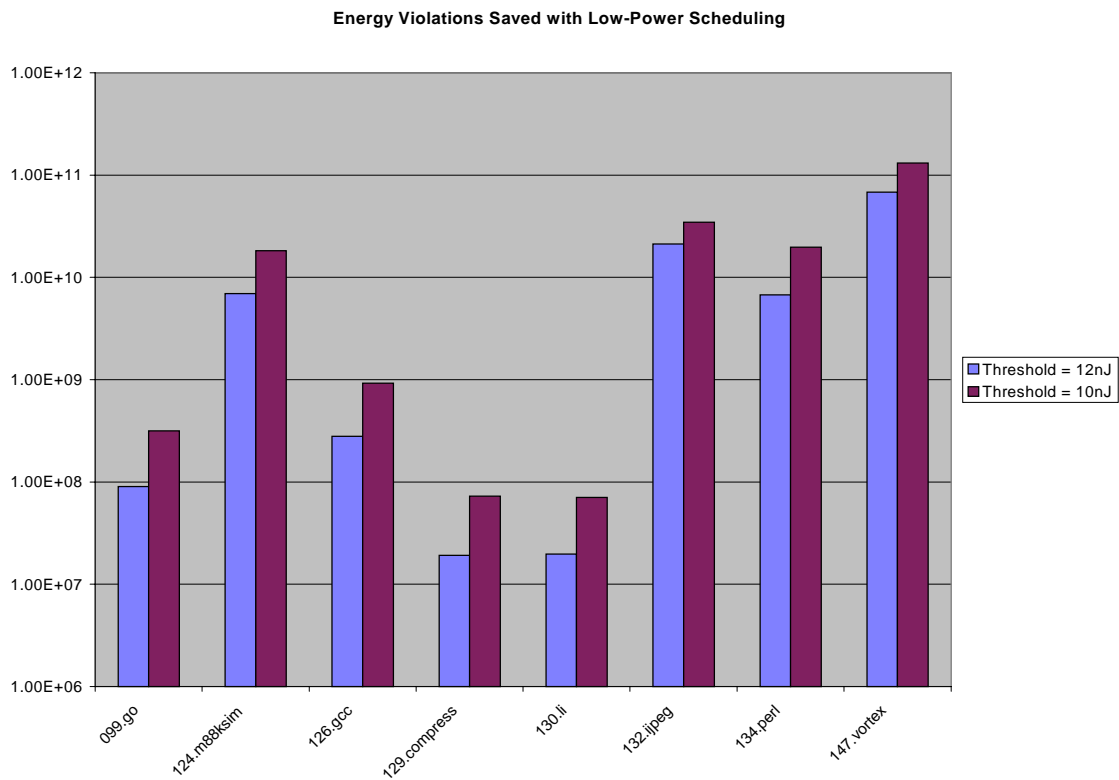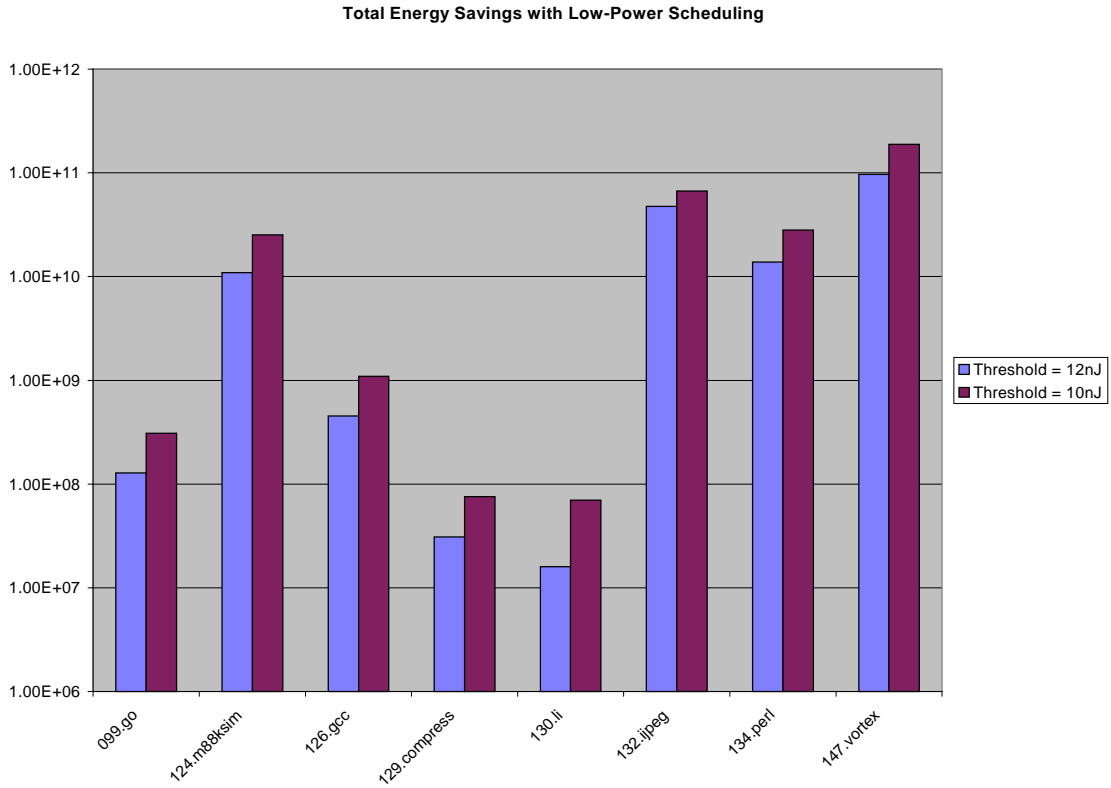
Figure 2

The reduction in savings in total energy violations and total energy is inversely proportional to the amount of performance improvement achieved by increasing the energy threshold. Figures 1 and 2 clearly demonstrate this relationship. For the 12nJ threshold, Figures 1 and 2 show that the amount of violations and total energy saved falls off a bit from the 10nJ threshold. However, we feel that this reduction is acceptable given the performance gain we achieve by increasing the threshold value.

In addition to the results presented above for total energy violations and energy savings, we measured the extra energy cost in each benchmark program for every 100 cycles scheduled. For every 100 cycle segment, we calculate the total amount of excess power dissipated. Table 1 shows that excessive power is typically in the range of 10.0 to 100.0 nJ for every 100 cycles. Over large programs, this results in a large amount of excessive power dissipation which can be saved using the low-power scheduling technique proposed here.

Table 1 – Excessive Power Distribution per 100 Cycles

	$0 < X < 1.0$	$1.0 < X < 10.0$	$10.0 < X < 100.0$	$100.0 < X$
099.go	1330	3272	2702	0
124.m88ksim	55	160	236	8
126.gcc	5752	17566	15870	255
129.compress	15	26	29	0
130.li	214	290	191	0
132.ijpeg	78	309	461	30
134.perl	754	1516	772	4
147.vortex	116	1764	6754	63

In Table 1, the data represents the number of times that the total power difference was in the specified range during 100 cycle spans.

4 Conclusions

In this paper, we have proposed a new method for scheduling instructions which helps reduce power dissipation in high performance processors. The approach is based on a per cycle energy threshold which may not be violated in any given cycle. Instructions are scheduled based on the list scheduling algorithm until the threshold for the current cycle is reached. Once the threshold has been exceeded, the scheduler begins scheduling for the next cycle. We have shown that this method can result in significant energy savings over a given program with little to no performance impact.

Future plans for this research are to extend the energy model to contain a more concise representation of the desired architecture and to investigate further enhancements to the scheduling algorithm to allow further increased savings.

5 References

[1] Su, C-L., Tsui, C-Y., and Despain, A.M., "Low Power Architecture and Compilation Techniques for High-Performance Processors," in *Proc. of the IEEE COMPCON*, pp.489-498, 1994.

[2] Tiwari, V., Malik, S., and Wolfe, A., "Compilation Techniques for Low Energy: An Overview," presented at the *1994 Symposium on Low-Power Electronics*.

[3] Tiwari, V., Malik, S., and Wolfe, A., "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," in *IEEE Trans. on Very Large Scale Integration Systems*, pp.437-445, 1994.

[4] Lee, M.T-C., Tiwari, V., Malik, S., and Fujita, M., "Power Analysis and Low-Power Scheduling Techniques for Embedded DSP Software," presented at the *1995 Int. Symposium on System Synthesis*.

[5] Hwu, W.W., Mahlke, S.A., Chen, W.Y., Chang, P.P, Warter, N.J., Bringman, R.A., Ouelette, R.G., Hank, R.E., Kiyohara, T., Haab, G.E., Holm, J.G., and Lavery, D.M., "The Superblock: An effective structure for VLIW and superscalar compilation," in *The Journal of Supercomputing*, vol. 7, pp.229-248, Jan. 1993.

[6] Mahlke, S.A., Lin, D.C., Chen, W.Y., Hank, R.E., and Bringman, R.A., "Effective compiler support for predicated execution using Hyperblock," in *Proc. of the 25th Ann. Int'l. Symposium on Microarchitecture*, pp45-54, 1992.

[7] Havanki, W.A., *Treegion scheduling for VLIW processors*. MS Thesis, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1997.

[8] Gyllenhaal, J.C., *A machine description language for compilation*. MS Thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1994.

[9] Gyllenhaal, J.C., Hwu, W.W., and Rau, B.R., "HMDES Version 2.0 Specification,", Technical Report IMPACT-96-3, University of Illinois at Urbana-Champaign, Urbana, IL, 1996.