TWO APPROACHES TO OPTIMIZING FOR POWER IN THE SPIRAL FRAMEWORK

Peter Milder and Marek Telgarsky

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

This paper evaluates two methods for generating power or energy optimized Digital Signal Processing (DSP) transforms. Both approaches integrate power models into the SPIRAL code generator. SPIRAL decomposes a transform into an algorithmic formula representation, which is then implemented in C and timed. The timing data is used as feedback to control the parameters of the next algorithm generated. By replacing the timing method with power estimation models, we direct SPIRAL to search for a power-optimized implementation.

We implement and evaluate two different methods: a processor simulator and power model, and a hardware based method that models power based upon data collected directly from on-chip Pentium 4 performance counters.

1. INTRODUCTION

A large amount of effort is directed into power concerns during the design of desktop and embedded processors, but very little work has been done examining power efficient software. Although hardware design is a very important factor in low-power systems, the software design may play a significant role in the consumption of power and energy.

1.1. Motivation

This work aims to analyze the effects of software implementation on processor power and energy consumption. The overarching goal of this research is to gain an understanding of the degree to which software implementations affect power issues.

Digital Signal Processing (DSP) transforms are often used in power-sensitive applications, and as such, are an excellent class of software to optimize for power and energy. The SPIRAL code generator [1] creates runtime-optimized DSP kernels by searching over many implementation degrees of freedom and timing the results for feedback. By integrating power modeling techniques as feedback mechanisms, SPIRAL can be used to generate power or energy-optimized software.

1.2. Previous Work and State-of-the-Art

This research is based upon the well-developed SPIRAL framework. SPIRAL uses its own signal processing language to mathematically express DSP transforms. By supporting several different decomposition rules it is capable of expressing a transform of a given size in many ways. While the algorithms all compute the same result, the differences in their structures result in a variety of runtimes. Being highly extensible, the SPIRAL platform's backend can be changed from runtime performance to power measurement.

We examine two methods for modeling power. The first model, the Wattch simulator, [2] is often used in power estimation. It is a pure software simulator of a hypothetical processor, with power consumption calculated as a linear combination of simulated event counters. The second method we examine is an empirical macro scale power model of the Intel Pentium 4 developed by Isci and Martonosi [3]. It is based on metrics unveiled by the performance counters available on the Pentium 4.

1.3. Overview

The SPIRAL measurement back-end is easily modifiable to optimize for various metrics. We modify this backend and insert, separately, the two different power models. Once the models are in place and stable, we concentrate on 2-power sized DFTs. SPIRAL then generates different algorithms for DFTs of a particular size and executes them under the two different models. Our primary goal is to evaluate these methods and their ability to generate power or energy-optimized transforms. Secondly, we hope to look for correlations between runtime, energy consumption, and power consumption in each model.

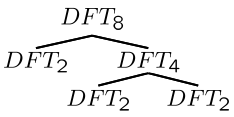
1.4. Organization of the Paper

We begin Section 2 by providing an explanation of the SPIRAL platform and why it is an appropriate testbed for measuring power usage in software. Next, we give details, including positives and negatives, of each of the two simulation methods. Section 3 concentrates on the actual implementation. Section 4 then gives the results and section 5

Transform DFT_n

Rule $DFT_{nm} \rightarrow (DFT_n \otimes I_m) \cdot D \cdot (I_n \otimes DFT_m) \cdot P$

Ruletree



Formula $DFT_8 = (F_2 \otimes I_4) \cdot D \cdot (I_2 \otimes (I_2 \otimes F_2 \cdots)) \cdot P$

Fig. 1. Using rules to decompose a transform into a rule tree and formula.

provides some conclusions and potential for future work.

2. NECESSARY BACKGROUND

2.1. SPIRAL

SPIRAL is a generator for platform-optimized DSP code. It takes a user-specified transform as input, and produces a runtime-optimized C implementation. As mentioned before, SPIRAL uses a signal processing language to mathematically represent a number of important transforms in signal processing. For our evaluations we concentrate on the Discrete Fourier Transform (DFT). Using the Cooley-Tukey decomposition rule, the DFT can be decomposed in a number of ways into smaller DFTs. By expressing Cooley-Tukey mathematically, SPIRAL can generate different ruletrees for each possible expansion at each DFT size. Graphically, this is represented in Figure 1. Each tree is converted into C code that is then run under a timing backend. In our work, we replace this timing backend with one of two power models. Figure 2 shows a block diagram of SPIRAL with our integrated power evaluation methods.

SPIRAL is the perfect platform for testing power usage because of its ability to generate different implementations of the same algorithm. By comparing the power used by each of the implementations, it is possible to show that some are better in terms of power or energy consumed than others. It is also very easy to compare these power figures to the number of clock cycles. Thus we can determine the power, energy, and runtime for each generated implementation.

2.2. Wattch

Wattch is an architectural-level power simulation tool based upon the SimpleScalar [4] cycle-level processor simulator. The Wattch model calculates baseline and per-access power costs for all of the simulated processor's functional units. Then, it uses event counters to determine the number of accesses for each unit. The total power and energy consumed by a program are modeled as linear combinations of the power and energy consumed in the individual functional units.

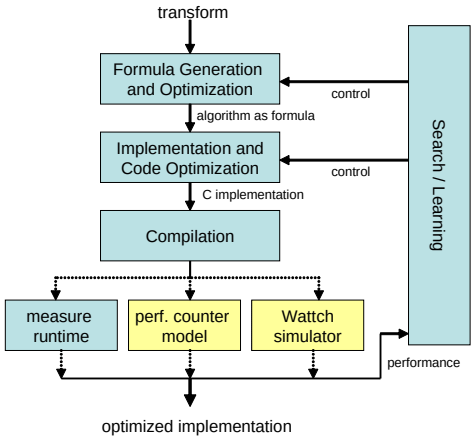


Fig. 2. Integrating power simulation into the SPIRAL environment

Although Wattch runs much faster than circuit or HDL-level power simulators, it is still quite slow for this application since, typically, SPIRAL generates and evaluates a large number of implementations. Compilation and simulation can take as long as several minutes for a medium-sized transform. In an application where thousands of large transforms need to be searched in order to find an optimal solution, the effectiveness of the search can be greatly limited by these time constraints.

2.3. Performance Counter Model

Isci and Martonosi produced a power model for the Pentium 4 processor based on performance counter metrics. This model, described in [5], is for a specific Pentium 4 die: the 0.18 micron 1.4 Ghz Willamette. The model subdivides the die into 22 units. The power used by each unit is then represented by a linear combination of performance counters. Using test programs that stressed different units while measuring current into the processor with an ammeter, Isci correlated the performance counter values to the required power. However, not all details of the model are available in [5], and personal correspondence [6] with the author was necessary to complete the model.

The performance counter model described in [3] uses a custom kernel module under Linux to access the performance counters on the Pentium 4. In this work, the freely available perfctr library is used [7]. Using a supported solution means it is portable between kernel revisions. Also, the library, being partially embedded in the kernel, turns off the counters when the process under test is not executing. This was not a concern in Isci's work as they were interested in system-wide power performance, while our concern is much more fine grained. The drawback of the library is that it introduces some overhead into the retrieval of the counter values. How much the measurement affects the test depends

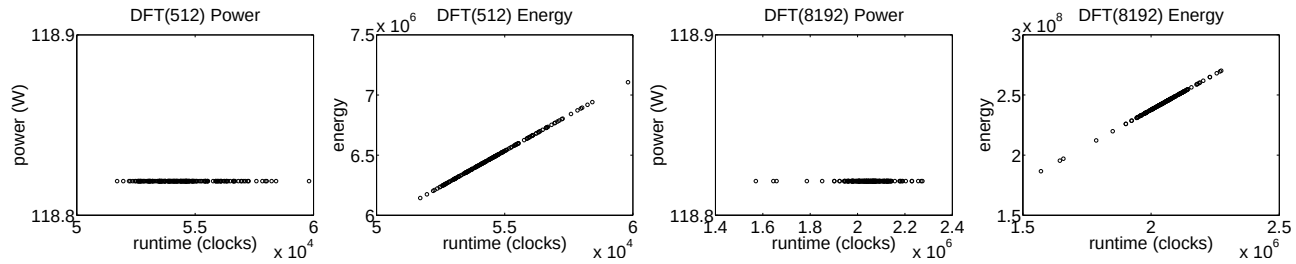


Fig. 3. Wattch simulated power and energy for random rule trees of DFT_{512} and DFT_{8192}

upon the counters under study. Section 3.2 describes some of the techniques employed to combat this problem.

3. IMPLEMENTATION DETAILS

We execute 200 random rule trees of DFTs of sizes 64, 512, 8192, and 32768. The transform sizes were chosen such that their data span a range of sizes ranging from residing entirely in the L1 cache to being larger than the L2.

3.1. Wattch

Wattch is configured to run an out-of-order core with a split L1 cache (16KB data and 32KB instruction) and a 256KB L2 cache. Code for the simulator is compiled with a version of gcc that has been modified to output PISA assembly, which is the instruction set architecture used by SimpleScalar. All compilation was done with the `-O3` flag.

3.2. Performance Counter Model

There are several issues with the implementation of the performance counter model. First, the model was tuned for the aforementioned 1.4Ghz CPU, while the system under test is a dual 1.7Ghz Xeon. The L1 and L2 cache sizes of the original CPU are not known, while the CPU under test has an 8KB L1 data cache, a 12KB L1 instruction cache, and a 256KB L2 cache. For the purpose of this work, we assume that the model holds even though the Xeon under test and the original Willamette are different.

A test consists of making four passes on each generated algorithm. The list of performance counters monitored is slightly different on each pass. The counters are then combined linearly and divided by the number of clock cycles to produce a power figure per unit. These power figures are then summed together into the final result. Sometimes the cycle count for one pass will be much longer than for the rest. To deal with the problem, each test is run 10 times. A test is dropped if any of its passes take longer than 5 percent of the median of all the passes in all the tests for that particular transform. This removes the occasional hiccup due to context switches and other runtime factors. The power figures for the remaining good tests are then averaged to yield the final estimate.

4. EXPERIMENTAL RESULTS

4.1. Wattch

Figure 3 shows scatter plots of the Wattch simulated power and energy consumption correlated with runtime for DFT_{512} and DFT_{8192}.

The Wattch simulator's results show a reasonable range and spread of energy values. There is approximately a factor of 3 difference between the highest and lowest energy in each of the four sizes examined. As expected, the energy consumed increases with an increase in transform size.

However, the results show that the energy consumed is linearly correlated with the algorithm's runtime. Accordingly, the results for simulated power show a constant level of power dissipation across all generated algorithms at all runtimes. These results show power as a constant value $\pm 1/100,000$ watt.

Although the same tests were run over algorithms of four different sized transforms, the results are the same for each: constant power and linearly increasing energy. For this reason, results for other sizes are not displayed.

The very narrow range of power suggests that Wattch (as currently configured) is not a reasonable method to optimize for power consumption within SPIRAL. Although there is a range of reported values, the spread is so narrow that it is impossible to tell if the results are significant at all, or if the variations are simply noise.

Furthermore, the near-constant power values show that the Wattch method is not ideal for optimizing for energy consumption in SPIRAL. Since the simulated power is extremely close to constant, the energy results reported by Wattch are extremely closely correlated to runtime. Therefore, optimizing for energy gives the same result as optimizing for runtime, but takes much longer to compute.

4.2. Performance Counter Model

In contrast to the Wattch model, the performance counter model results in a significant spread of results for each size under test (see Figure 4). The first case, DFT_{64}, fits entirely into the L1 data cache and produces an unexpected result that does not correlate with the rest of the sizes, namely, that there are implementations that consume the least power

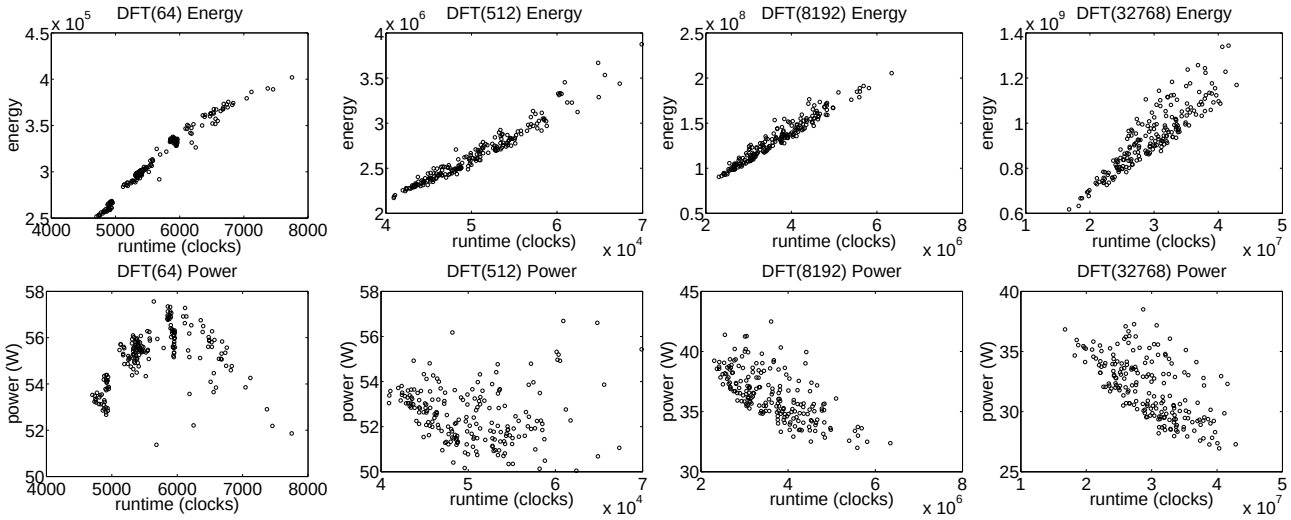


Fig. 4. Performance counter model power and energy for random rule trees of $\mathrm{DFT}(64, 512, 8192, \text{and } 32768)$.

and have the shortest runtime. It is unclear why this would be the case, but it is possible that the overhead of reading the counters is affecting the measurements. A DFT_{512} fits entirely into the L2 cache, and exhibits the expected behavior with power inversely proportional to runtime. The last two sizes do not fit into cache and exhibit behavior similar to the 512 case. In both of the largest cases the inverse relationship between power and runtime is clear. For the fastest algorithms of each size, the shortest runtime leads to the least energy consumed. Additionally, it is possible to choose a best runtime given a power constraint. The range of the results suggests that there are differences between implementations and that this model may be effective in categorizing them.

5. CONCLUSIONS

We have presented two methodologies for measuring transform power in SPIRAL. The SPIRAL project has shown that given a transform of some size, different algorithms produce different runtimes. By replacing the SPIRAL measurement backend, we have shown that an algorithm's energy and power needs, like runtime, are dependent on its structure.

The Wattch simulator results for power have been inconclusive in that they suggest all algorithms consume the same amount of power. Given the variance in runtimes, it is unlikely that this is an accurate simulation. The performance counter model provides reasonable results, although its absolute correctness has not been verified.

Future work includes further examination of the Wattch simulation to determine why large changes in runtime and cache hit rates do not lead to changes in power consumed. Additionally, the performance counter model needs to be fully justified on the machine under test.

6. REFERENCES

[1] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. Johnson, and N. Rizzolo. SPIRAL: Code Generation for DSP Transforms. In *Proceedings of the IEEE special issue on "Program Generation, Optimization, and Adaptation,"* Vol. 93, No. 2, 2005, pp. 232-275.

[2] D. Brooks, V. Tiwari, M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*, 2000.

[3] C. Isci, M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 93-104.

[4] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. *Computer Archictecture News*, pages 13-25, June 1997.

[5] C. Isci, M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. Internal Princeton University Dept. of Electrical Engineering Technical Report, 2003.

[6] C. Isci. Personal communication. 17 April 2005.

[7] M. Pettersson. Perfctr library for Linux. http://user.it.uu.se/ mikpe/linux/perfctr/. Accessed 18 April 2005.