Profile-Driven Code Execution for Low Power Dissipation

Diana Marculescu
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract - *This paper proposes a novel technique for power-performance trade-off based on a profile-driven code execution methodology. Specifically, we show that there is an optimal level of parallelism for energy consumption and propose a compiler-assisted technique for code annotation that can be used at run-time to adaptively trade-off power and performance. As shown by experimental results, our approach is up to 23% better than clock throttling and is as efficient as voltage scaling (up to 10% better in some cases). The technique proposed in this paper can be used by an ACPI-compliant power manager for prolonging battery life or as a passive cooling feature for thermal management.*

1 Introduction

Power dissipation has become a critical design concern in recent years, driven by the increased levels of complexity and emergence of mobile applications. While it is generally agreed that tools for power estimation and optimization do exist for hardware specifications at different levels (circuit, gate, register-transfer or behavioral), more work is needed in the area of power analysis or optimization at microarchitecture, architecture or system level [1]. 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.

In the area of power modeling for embedded software, [2] proposes a *per-instruction base power model* that can be used to find an aggregate power estimate for a sequence of instructions. In [3], the case of DSP applications is addressed. There, the inter-instruction effects turn out to be significant, thus making possible to develop instruction scheduling techniques that target power minimization. The authors of [4] present an architectural enhancement to reduce the extra work or energy due to mispredicted branches, without significant loss in performance. In [5] a technique for reducing the average power consumption for the pipeline structure is presented. Other approaches target techniques for energy efficient memory systems [6,7]. From a different perspective, the aspect of thermal management has been addressed in [8] where a hardware-driven technique for *instruction cache throttling* has been proposed.

In this paper we address the problem of energy optimization in modern processors by using compiler-assisted code annotation for *variable fetch* or *execution rate*. We improve the state-of-the-art by proposing a novel technique for *fine-grain* energy characterization based on a *profile-driven* code execution methodology. Specifically, we show analytically and experimentally that there exists an optimal level of parallelism for energy consumption (which may not be necessarily the same as the one for performance) and propose a *compiler-assisted* technique for code annotation that adaptively selects at run-time the optimal number of instructions to be fetched or executed in parallel as far as energy is concerned. Energy, as opposed to performance, is a much more data-dependent parameter. As it will be shown subsequently, it is indeed possible to use less than the maximum number of functional units available, and achieve less energy consumption. We study this effect for the execution stage, as well as for the entire processor. For the first time to our knowledge, we show that there exists an inherent trade-off between *performance* and *energy consumption*, due to the data-dependency effect, but, most importantly, due to *speculative execution* and the inherent *level of parallelism* exhibited by common applications. To validate our results, we use a microarchitecture-level power simulator developed in industry [9]. As shown subsequently, significant savings can be obtained in both energy and power consumption, at the expense of some decrease in performance.

The techniques described in this paper can be used as a means for prolonging the *battery life*, but most importantly, for *thermal management* [10] by achieving significant average power reductions in the execution stage or throughout the chip. Such an approach could be used in the context of power management schemes using the Advanced Configuration and Power Interface (ACPI). Thermal management in ACPI is achieved via average power reduction through "clock throttling" in the case of *passive cooling*, or by turning on the on-chip fan in the case of *active cooling*. However, both techniques are likely to actually decrease the battery life by consuming more energy, although the average power per cycle is decreased. Our proposed technique is up to 20% more efficient than the classic clock throttling technique and also reduces the total energy consumed, thus prolonging battery life as a by-product. Also, as opposed to the instruction cache throttling technique presented in [8], our characterization is *software-driven* and provides a fine grain energy characterization on a *per basic block, per process* basis.

The paper is organized as follows: Section 2 presents the rationale behind profile-driven code execution. In Section 3, we present our proposed methodology for code-annotation for low energy code execution. Section 4 shows our experimental results on a subset of SpecInt95 benchmarks. We conclude in Section 5 with some final remarks.

2 Profile-Driven Instruction Execution for Energy Optimization

In superscalar processors, the hardware may execute from one to eight instructions per cycle. Usually, these instructions must be independent and satisfy some constraints. If a certain instruction is dependent or doesn't meet the constraints, only the instructions preceding it in the sequence are issued, hence the variability in issue rate. If, in addition, out-of-order execution of instructions is permitted, any of the succeeding instructions may be executed if there are no data dependencies present. Let us consider the simple case of an in-order execution of a computation-intensive piece of code. We show subsequently how the total energy per operation is decreased by trading-off performance for power.

Example 1. Consider the computation of the product of two input streams[1] and the availability of up to four 16-bit multipliers able to perform the multiplication. We consider four possible scenarios corresponding to using one, two, three or all of the functional units. The four scenarios, labeled with the corresponding *total energy per operation*, are depicted in Fig.1. (It is assumed that the input stream x_i includes both operands.) As it can be seen, the energy consumption per operation varies significantly among the four scenarios. This is mainly due to the very different profile of the data that is sent to each of the available functional units. Also, when comparing the performance, we can see that using four multipliers gives the largest performance (4 computations per clock cycle), but using two multipliers reduces the total energy by more than 32% and average power by 65%, when compared to the worst case.

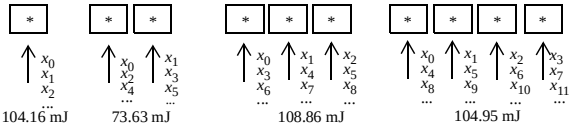

Fig.1 Four ways of computing a series of multiplications

This type of behavior is actually found very frequently in practice, especially in computation intensive applications. These effects become even more important in the case of DSP processors where most of the power consumption is due to the datapath performing additions or multiplications. Another scenario where this effect may become prevalent is the one of a mix of load/store and arithmetic instructions that alternatively use the same functional units for computing effective addresses and output values. Typically, memory addresses look much more "different" than the operands of arithmetic instructions and thus, a behavior similar to the one shown above will arise.

However, in superscalar processors, the contribution of the datapath (i.e., execution stage) to the total energy consumption may not be impressive. More precisely, the fetch and issue stages which actually schedule and dispatch instructions to the execution stage have a significant contribution to the total energy consumption. A typical pipeline structure for an out-of-order superscalar processor is shown in Fig.2. The fetch stage can bring a fixed number of instructions from the I-cache, while in the dispatch stage, the instruction decoding and register

1. We have used a real data stream from a DSP application.

renaming is performed. The scheduler tracks memory and register dependencies and issues as many instructions as possible to the execution stage. Not surprisingly, whether or not the execution stage does any useful computation, the fetch, dispatch and schedule stages are always active for the purpose of increasing performance via fetching multiple instructions, register renaming, resolving dependencies and doing out-of-order issue to the execution stage.

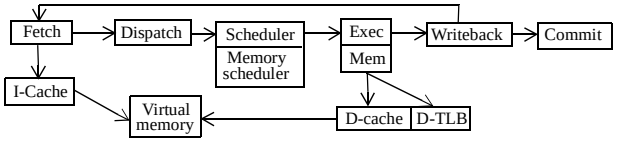

Fig.2 The pipeline of a typical superscalar processor [12]

In what follows, we will present the analytic model which allows for reducing total energy of the execution stage in datapath dominated architectures (VLIW or application specific architectures). In addition, we will discuss the motivation for varying the fetch and execution rates in the case of control dominated architectures (superscalar architectures, with out-of-order and speculative execution).

2.1 Datapath dominated architectures

This section presents theoretical evidence in support of the inherent energy-performance trade-off shown in Example 1. It mainly targets architectures that are datapath dominated (such as VLIW or DSP), although the results are valid for superscalar processors as well.

Power consumption is a function of the switched capacitance of a module and there is a close relationship between the power consumption of a functional unit and the number of transitions on its primary inputs. Thus, a simple and effective way to characterize the effect of an input sequence on power consumption is via its average Hamming distance. The following result provides a characterization of the average Hamming distance between vectors that are separated by an arbitrary (but fixed) number of time steps. The input stream is assumed to be modelled by a lag-one Markov chain. (All proofs can be found in [11,13].)

Proposition 1. If an input sequence $\{x_n\}_{n \geq 0}$ is modeled by a lag-one Markov chain, $p(x)$ is the probability of occurrence of vector x and $p_k(x|y)$ is the conditional probability of making a transition from y to x in exactly k steps, then the average Hamming k-distance denoted by $d_k = \sum_{x,y} p(y) \cdot p_k(x|y) \cdot d(x,y)$ converges when $k \to \infty$ and the limit is $D = \lim_{k \to \infty} d_k = \sum_{x,y} p(x) \cdot p(y) \cdot d(x,y)$.

d_k is the average Hamming distance between input vectors that are exactly k steps apart. Thus, d_k is a measure of the average switching activity when exactly k units are used to process the input sequence in parallel. Although the above result is valid only for finite-order Markov chains, we should point out that in practice, data traces or memory references do exhibit a finite time interval in which dependencies are present (due to *spatial* and *temporal locality*).

This result can be extended to any finite-order Markov chain and to other measures that may be relevant to the actual power dissipation, such as signal probabilities or number of ones in the input sequence. In addition, since most power macromodels are based on switching activity and other input statistics that satisfy similar properties as the ones in Proposition 1, the average energy per operation will show a similar trend. More precisely, the following result holds for the energy per operation of a given type of functional unit:

Proposition 2. If a data trace $\{x_n\}_{n \geq 0}$ is modeled by a lag-one Markov chain, and $E_k = E_m(d_k)$ is the energy consumed per operation when k identical functional units are available, then $E = \lim_{k \to \infty} E_k = E_m(D)$ where E_m is the energy macromodel and D is as in Proposition 1.

The energy cost per operation for k instructions issued in parallel is proportional to d_k and from the above result, it can be seen that *no gain* can be achieved by increasing the parallelism over some limit. Moreover, the optimal solution (that is, the value of k for which d_k is minimized) is found for *small* values of k. Typically, d_k has the behavior depicted in Fig.3, with a rapid convergence to the limit after k reaches a value of five or six. The minimum energy can be obtained for any value of k, including 1.

In practice, since the level of parallelism in most user programs is limited, to find the value of k for which the total energy per operation (or d_k) is minimized, we only need to consider a finite number of configurations. We point out that in the context of out-of-order or speculative execution, the actual data stream looks different than in the case of in-order execution, but the behavior is similar to the one in Fig.3.

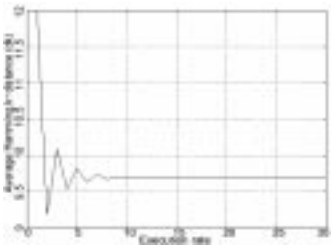

Fig.3 Average Hamming k-distance vs. k (typical example)

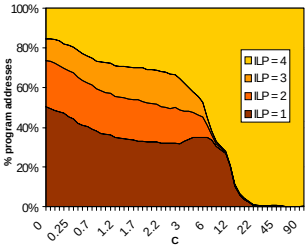

Fig.4 ILP for optimal $E*D^C$ (*compress95*)

2.2 Control dominated architectures

The results presented in Section 2.1 are valid for the characterization of datapath dominated architectures, without any data dependencies. In modern processors, extensive use of out-of-order execution makse the contribution of datapath to the total energy cost less important. In addition, modern processors use a fair amount of branch prediction and speculation. However, this does not come without a price in the amount of energy consumed, especially when a branch is mispredicted. Thus, as the execution width is increased from 1 to 2, 3, or more, the amount of unnecessary energy consumed due to mispredictions is likely to increase.

Another factor that is important in how the resources of the processors are used is the *instruction-level parallelism* (ILP) exhibited by the user code. Depending on the inherent level of parallelism exhibited by common applications, the overhead due to the dispatch and schedule stages will increase with increasing fetch and issue rates, although the same amount of parallelism is uncovered. For a piece of code that is less parallelizable, fetching the instructions slower will decrease the extra power consumed, as well as the energy cost due to mispredictions. On the other hand, if more instructions than necessary are fecthed, the fetch queue will fill up quickly and instructions will wait there for a long time until they are sent to the next stage of the pipeline. To illustrate this behavior, we show in Fig.4 the distribution of ILP for optimal $E*D^C$ (E=energy, D=cycles). While ILP has to be 4 for achieveing high performance (right side of the chart), if the target is low energy (or other variants), the ILP is almost equally distributed among all values 1-4 (left side of the chart).

All these issues make it difficult to develop an analytic model that is able to unravel all power-performance trade-offs present in common applications. To overcome these limitations, we need to find mechanisms able to finely tune the fetch, issue and execution rates according to the actual program characteristics. For this purpose, we propose a profile-driven methodology, as described in the following section.

3 Profile-Driven Code-Annotation for Low-Energy Code Execution

Based on the results presented in Section 2, a simple profile-driven methodology can be used for finding the optimal number of instructions to be executed in parallel for a given *basic block* (a basic block is a straight line code sequence with no transfers in or out, except at the beginning or the end). What we target is a *fine grain* characterization of the basic blocks encountered in the typical execution of a program as far as the optimal number of instructions to be executed in parallel is concerned. Typical basic blocks that will benefit from such a methodology are loops, unrolled to uncover more parallelism.

To capture both the data dependency effect and, more importantly, the effect of out-of-order and speculative execution, we propose a *profile-driven methodology* to find the optimal number of instructions to be executed in parallel for each basic block. We first simulate the input program for a typical input stream K times, varying the fetch or execution rate between 1 and K (the maximum available rate for the architecture under consideration). Then, after collecting the energy values for each basic block, the value of k which gives the minimum value of the energy consumption for the entire system is used to annotate all instructions in the basic block under consideration. This assumes that the instruction format allows for recording this information and that the microarchitecture of the processor is slightly modified to support variable fetch or execution rates.

4 Experimental Results

We have implemented the methodology described in Section 3 by using the *sim-outorder* simulator from the SimpleScalar suite [12]. Its instruction format allows for code annotation with the rate to be used at run-time for fetch or execution. To validate our proposed techniques, each basic block is annotated with the level of parallelism which gives

Table 1: Fixed and variable *execution rate* for SpecInt'95 benchmarks

Benchmark	Exec. rate = 1			Exec. rate = 2			Exec. rate = 3			Adaptive (unconstr.)					Adaptive (constr.)				
	P	E	C	P	E	C	P	E	C	P	E	C	P_{CT}	E_{VS}	P	E	C	P_{CT}	E_{VS}
compress95	0.62	0.85	1.37	0.90	0.95	1.06	0.99	0.99	1.00	0.67	0.81	1.21	0.83	0.83	0.74	0.90	1.11	0.90	0.91
li	0.51	0.85	1.68	0.87	0.94	1.08	0.98	1.01	1.03	0.59	0.79	1.33	0.75	0.76	0.69	0.82	1.19	0.84	0.84
ijpeg	0.32	0.89	2.81	0.83	1.08	1.29	0.96	0.96	1.00	0.33	0.77	2.33	0.43	0.51	0.49	0.91	1.85	0.54	0.59
m88ksim	0.55	0.92	1.68	0.93	1.02	1.09	1.04	1.04	1.00	0.68	0.89	1.31	0.76	0.76	0.76	0.88	1.15	0.87	0.88
Average	0.50	0.87	1.88	0.88	0.99	1.13	0.99	1.00	1.01	**0.56**	**0.81**	1.54	**0.65**	**0.71**	**0.67**	**0.87**	1.35	**0.74**	**0.81**

Table 2: Fixed and variable *fetch rate* for SpecInt'95 benchmarks

Benchmark	Fetch rate = 1			Fetch rate = 2			Fetch rate = 3			Adaptive				
	P	E	C	P	E	C	P	E	C	P	E	C	P_{CT}	E_{VS}
compress95	0.63	0.88	1.40	0.87	0.96	1.10	0.98	0.99	1.01	0.65	0.84	1.28	0.78	0.78
li	0.54	0.95	1.76	0.83	0.93	1.12	0.96	0.97	1.01	0.80	0.90	1.12	0.89	0.91
ijpeg	0.41	1.11	2.68	0.69	1.03	1.49	0.96	0.99	1.04	0.99	0.99	1.00	1.00	1.00
m88ksim	0.54	0.97	1.80	0.89	0.94	1.05	0.98	0.96	0.98	0.86	0.89	1.03	0.97	0.98
Average	0.53	0.98	1.91	0.82	0.96	1.19	0.97	0.98	1.01	**0.85**	**0.92**	1.07	**0.93**	**0.92**

the *minimum energy consumption*. We note that other criteria can be used for optimization, such as energy delay product, or energy optimization with performance constraints, etc. We have implemented the *meta-block* concept in which each basic block is assigned the same execution or fetch rate as its neighbors, when they execute in sequence most of the time [11,13]. To validate the results, the annotated code is then run through a modified version of *sim-outorder* which allows for dynamic selection of the execution or fetch width according to the information stored in the annote field. To this end, we performed two sets of experiments.

A. Datapath dominated architectures: adaptive *execution* rate for reducing the energy consumption of the execution stage. For this purpose, we have augmented the *sim-outorder* simulator with information about the power consumption of the datapath only. The datapath modules are pre-characterized with power macromodels that are instantiated with actual profile data.

We report in Table 1 our results for some of the SpecInt95 benchmarks. In columns 2-10 we present the effect of executing less than 4 instructions each clock cycle. All values are normalized with respect to the base case of 4 instructions executed every cycle. As we can see, decreasing the level of parallelism decreases performance (C) in all cases, compared to the base case. On the other hand, total energy (E) increases steadily with the level of parallelism for *compress95*, while for the other three benchmarks, the worst energy case is obtained either for 3 (*li* and *m88ksim*) or 2 (*ijpeg*) instructions executed every cycle. In addition, the average power (P) varies a lot among the 4 scenarios and it can be about 68% less (for *ijpeg*) in the case of one instruction executed per cycle when compared to the base case.

In columns 11-20 we present the effect of adaptively selecting the optimal execution rate for each basic block. In the constrained case, we selected the number of instructions to be executed in parallel which minimizes the total energy per basic block, but doesn't increase the number of cycles by more than twice. For comparison, all results are normalized with respect to the base case of up to 4 instructions executed each cycle. We also report the normalized values for average power values obtained using clock throttling (P_{CT}) and total energy when using voltage scaling (E_{VS}), both producing the same performance penalty as our adaptive execution rate technique. In the unconstrained case we get an energy savings 19% on average. In addition, an average power savings of 44% can be obtained with about 35% decrease in performance[2], while the improvement over clock throttling can reach 23% (for *ijpeg*) or almost 19% on average. Also, our technique performs better (*compress95*), or about the same (*li, m88ksim*) when compared to voltage scaling. In the constrained case, the total energy savings is about 13% on average, while the power savings is 33% on average, at the expense of 25% decrease in performance. The variable execution rate technique performs on average about 12% better than clock throttling and is about as efficient as voltage scaling. Our technique, however, does not require additional circuitry like voltage scaling does (DC-DC converters, level converters or variable clock frequency).

We have also studied the effect of changing the input files used for running each of the benchmarks. Our analysis [13] shows that the level of parallelism needed to reduce the total energy is not necessarily related to the actual data values fed to the execution stage and thus the annotation process is robust.

B. Control dominated architectures: adaptive *fetch* rate for reducing the energy consumption of the whole chip. To validate the results, we have used the same experimental setup as in *A.* However, in this case we have used an industry developed simulator [9] based on SimpleScalar which uses real power density values for each module of the processor [14]. The power model used is activity based and assumes a constant energy cost per module whenever that module is in use, and a non-zero (but much smaller) cost whenever is idle. Each basic block is characterized in terms of total energy for fetch rates of 1-4 and is annotated with the rate giving the minimum energy consumption. We report in Table 2 our results for full chip normalized values of average power, energy and number of cycles (P, E, C). Columns 2-10 present the case of fixed fetch rateand all values are normalized with respect to the base case (4 instructions fetched every cycle). In this case, the total energy, as well as the number of cycles vary significantly from one application to another.

In contrast, in columns 11-15 we report the results for adaptive fetch rate, on a per basic-block basis. As we can see, the total energy cost is reduced by up to 16% for *compress95* or 10% in the case of *li*. In the case of *ijpeg*, the savings are not impressive and this is mainly due to its inherent increased level of parallelism compared to the other benchmarks. We also report the power and energy values obtained when using clock throttling (P_{CT}) and voltage scaling (E_{VS}), respectively. As it can be seen, our adaptive approach can be up to 16% better than clock throttling and up to 10% better than voltage scaling. The average power consumption values are reduced by 15% on average, with a performance penalty of 6%. We note that up to 10-15% reduction in the average power and energy costs can be expected if a data dependent model is used.

To conclude, our approach can be efficiently used for trading-off performance for a decrease in both total energy and average power consumption. While the first will affect the battery life, the second can be used as an efficient means to reduce the operating temperature in a system equipped with a smart thermal manager. The annotated code can be used whenever the operating temperature increases over a threshold limit as a *passive cooling* technique, similar to the "clock throttling" state of the CPU in the ACPI specification [10]. However, for the same performance penalty, our approach is up to 23% better than clock throttling and can be up to 10% better than voltage scaling.

5 Conclusion

In this paper, we presented a novel technique for code optimization for low power based on a *profile-driven* methodology. Specifically, we show analytically that there is an optimal level of parallelism for energy consumption (which may not be necessarily the same as for performance) and we propose a *compiler-assisted* technique for code annotation that adaptively selects at run-time the optimal number of instructions to be fetched or executed in parallel as far as total energy is concerned.

6 Acknowledgments

The author would like to thank V. Tiwari, G. Cai and C. H. Lim of Intel, for useful comments and providing the power simulator, and S. Haga of UMD for coding a preliminary version of the approach.

7 References

[1] 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.

[2] 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.

[3] 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.

[9] 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*, June 1998.

[4] 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.

[5] 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,' to appear in *IEEE Transactions on VLSI Systems.*

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

[7] L. Benini, A. Macii, E. Macii, and M. Poncino, 'Selective Instruction Compression for Memory Energy Reduction in Embedded Systems,' in *Proc. ACM Intl. Symposium on Low Power Electronics and Design*, pp.206-211, Aug.1999.

[8] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, J. Alvarez, 'Thermal Management System for High Performance PowerPCTM Microprocessors,' in *IEEE COMPCON*, 1997.

[9] G. Cai and C.H. Lim, 'Architectural Level Power/Performance Optimization and Dynamic Power Estimation,' in *Proc. MICRO-32 (Cool Chips tutorial)*, Nov. 1999.

[10] 'Advanced Configuration and Power Interface Specification,' Intel, Microsoft, Toshiba, Revision 1.0b, Feb. 2, 1999, at *http://www.teleport.com/~acpi/DOWNLOADS/ACPIspec10b.pdf.*

[11] D. Marculescu and S.W. Haga, 'Adaptive Execution Rate for Low Power in Superscalar Processors,' Technical Report 99-10, Dept. of ECE, UMD, Oct. 1999.

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

[13] D. Marculescu, 'ILP Exploration for Energy Optimization in Modern Processors,' CMU-CEDA Technical Report, Feb.2000.

[14] G. Cai, Personal communication, Feb. 2000.

2. Performance is defined as the inverse of number of cycles.