Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower

N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye
Microsystems Design Lab
The Pennsylvania State University
University Park, PA 16802
(vijay,kandemir,mji,hykim,wye)@cse.psu.edu

ABSTRACT

With the emergence of a plethora of embedded and portable applications, energy dissipation has joined throughput, area, and accuracy/precision as a major design constraint. Thus, designers must be concerned with both optimizing and estimating the energy consumption of circuits, architectures, and software. Most of the research in energy optimization and/or estimation has focused on single components of the system and has not looked across the interacting spectrum of the hardware and software. The novelty of our new energy estimation framework, *SimplePower*, is that it evaluates the energy considering the system as a whole rather than just as a sum of parts, and that it concurrently supports both compiler and architectural experimentation.

We present the design and use of the *SimplePower* framework that includes a transition-sensitive, cycle-accurate datapath energy model that interfaces with analytical and transition sensitive energy models for the memory and bus subsystems, respectively. We analyzed the energy consumption of ten codes from the multidimensional array domain, a domain that is important for embedded video and signal processing systems, after applying different compiler and architectural optimizations. Our experiments demonstrate that early estimates from the *SimplePower* energy estimation framework can help identify the system energy hotspots and enable architects and compiler designers to focus their efforts on these areas.

Keywords: System Energy, Low-power Architectures, Compiler Optimizations, Energy Simulator, Hardware-Software Interaction, Energy Optimization and Estimation.

1. INTRODUCTION

With more than 95% of current microprocessors going into embedded systems, the need for low power design has become vital. Even in environments not limited by battery life, power has become a major constraint due to concerns about circuit reliability and packaging costs. The increasing need for low power systems has motivated a large body of research on low power processors. Most of this research, however, focuses on reducing the energy[1] in isolated subsystems (e.g. the processor core, the on-chip memory, etc.) rather than the system as a whole [7]. The focus of our research is to provide insight into the energy hotspots in the system and to evaluate the implications of applying a combination of architectural and software optimizations on the overall energy consumption.

In order to perform this research, architectural-level power estimation tools that provide a fast evaluation of the energy impact of various optimizations early in the design cycle are essential [2]. However, only prototype research tools and methodologies exist to support such high-level estimation. In this paper, we present the design of an architectural-level energy estimation framework, *SimplePower*. To our knowledge, this is the first framework with a capability to evaluate the integrated impact of hardware and software optimizations on the overall system energy. In contrast to coarse grain current measurement-based techniques [26; 17], our new tool is cycle-accurate, and provides a fine-grained energy consumption estimate of the processor core (currently a five-stage pipelined instruction set architecture (ISA)) while also accounting for the energy consumed by the memory and bus subsystems. *SimplePower* also leverages from the SimpleScalar toolset [3] as it is executes the integer subset of SimpleScalar ISA.

The memory subsystem is the dominant source of power dissipation in various video and signal processing embedded systems [6]. Existing low power work has focused on addressing this problem through the design of energy efficient memory architectures and power-aware software [12; 25; 23]. However, most of these efforts do not study the influence on the energy consumption of the other system components and even fewer consider the integrated impact of the hardware and software optimizations. It is important to evaluate the influence of optimizations on the overall system energy savings and the power distribution across different components of the system. Such a study can help identify the changes

[1]The dynamic energy consumed by CMOS circuits is given by $E = sCV^2$ where s is the switching activity on the lines, C is the capacitive load and V is the supply voltage. We do not consider the impact of leakage power.

ISCA 00 Vancouver, British Columbia Canada

in system energy hotspots and enable the architects and compiler designers to focus their efforts on addressing these areas.

This study embarks on this ambitious goal, specifically trying to answer the following questions:

• What is the energy consumed across the different parts of the system? Is it possible to evaluate this energy distribution in a fast and accurate fashion for different applications?

• What is the effect of the state-of-the-art performance-oriented compiler optimizations on the overall system energy consumption and on each individual system component? Does the application of these optimizations cause a change in the energy hotspot of the system?

• What is the impact of power and performance-oriented memory system modifications on the energy consumption? How do compiler optimizations influence the effectiveness of these modifications?

• What is the impact of advances in process technology on the energy breakdown of the system? Can emerging new technologies (e.g., embedded DRAMs [22]) result in major paradigm shifts in the focus of architects and compiler writers?

To our knowledge, there has been no prior effort that has extensively studied all these issues in a unified framework for the entire system. This paper sets out to answer some of the above questions using codes drawn from the multi-dimensional array domain, a domain that is important for signal and video processing embedded systems.

The rest of this paper is organized as follows. The next section presents the design of our energy estimation framework, *SimplePower*. Section 3 presents the distribution of energy across the different system components using a set of benchmark codes. The influence of performance-oriented compiler optimizations on system energy is examined in Section 4. Section 5 investigates the influence of energy-efficient cache architectures on system power. Section 6 studies the implications of emerging memory technologies on system energy. Finally, Section 7 summarizes the contributions of this work and outlines directions for future research.

2. SIMPLEPOWER: AN ENERGY ESTIMATION FRAMEWORK

Answering the questions posed in Section 1 requires tools that allow the architect and compiler writer to estimate the energy consumed by the system. The energy estimation framework that we have developed for this purpose, *SimplePower*, is depicted in Figure 1. For the purposes of this work, we are using a system consisting of the processor core, on-chip instruction and data caches, off-chip memory, and the interconnect buses between the core and the caches and between the caches and the off-chip memory. What we need in our framework are tools that allow us to estimate the energy consumed by each of the modules in the system.

Analytical models for memory components have been used successfully by several researchers [13; 25] to study the power

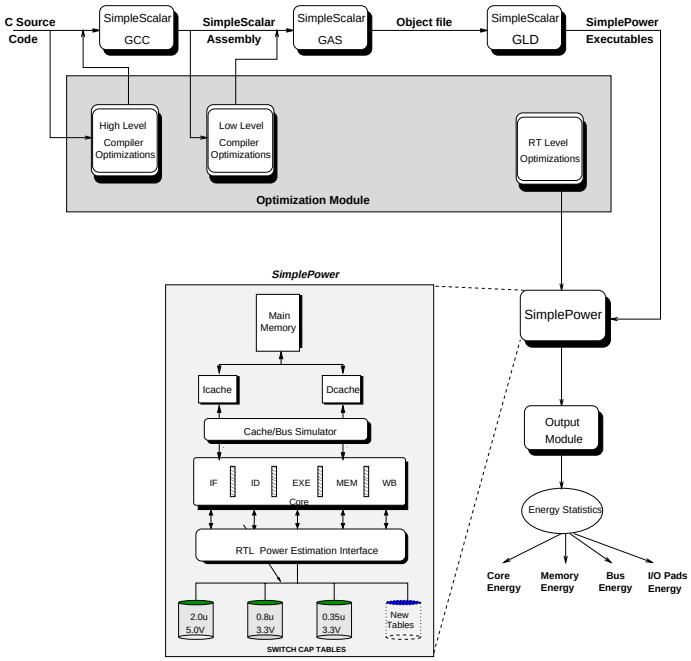

Figure 1: *SimplePower* energy estimation framework. It consists of the compilation framework and the energy simulator that captures the energy consumed by a five-stage pipeline instruction set architecture, the memory system and the buses.

tradeoffs of different cache/memory configurations. These models attempt to capture analytically the energy consumed by the memory address decoder(s), the memory core, the read/write circuitry, sense amplifiers, and cache tag match logic. Some of these models can also accommodate low power cache and memory optimizations such as cache block buffering [13], cache subbanking [25; 13], bit-line segmentation [12], etc. These analytical models estimate the energy consumed per access, but do not accommodate the energy differences found in sequences of accesses. For example, since energy consumption is impacted by *switching activity*, two sequential memory accesses may exhibit different address decoder energy consumption. However, simple analytical energy models for memories have proved to be quite reliable [13]. This is the approach used in *SimplePower* to estimate the energy consumed in the memories.

The energy consumption of the buses depends on the switching activity on the bus lines and the interconnect capacitance of the bus lines (with off-chip buses having much larger capacitive loads than on-chip buses). When the switching activity is captured by the energy model, we refer to the technique as a transition-sensitive approach (in contrast to, for example, the analytical model used for the memory subsystem). The energy model used by *SimplePower* for system buses is transition-sensitive. A wide variety of techniques have been proposed to reduce system level interconnect energy ranging from circuit level optimization such as using low-swing or charge recovery buses, to architectural level optimizations such as using segmented buses, to algorithmic level optimizations such as using signal encoding (encoding the data in such a way as to reduce the switching

activity on the buses) [11]. As technology scales into the deep sub-micron, chip sizes grow, and multiprocessor chip architectures become the norm, system level interconnect structures will account for a larger and larger portion of the chip energy and delay. In this paper, we include the energy consumed in the buses in the memory system energy, unless specified otherwise.

The final system module to be considered is the processor core. To support the architecture and compiler optimization research posed in Section 1, the energy estimation of the core must be transition-sensitive. At this point in the design process, in order to support "what-if" experimentation, the processor core is specified only at the architectural-level (RTL level). However, without the structural capacitance information that is part of a gate-level design description (obtained via time consuming logic synthesis) and the interconnect capacitance information that is part of a physical-level design description (obtained via very time consuming VLSI design), it is difficult obtain the capacitance values needed to estimate energy consumption. *SimplePower* solves this dilemma by using predefined, transition-sensitive models for each functional unit to estimate the energy consumption of the datapath. This approach was first proposed by Mehta, Irwin and Owens [20]. These transition-sensitive models contain switch capacitances for a functional unit for each input transition obtained from VLSI layouts and extensive HSPICE simulation. Once the functional unit models have been built, they can be reused for many different architectural configurations. *SimplePower* is, at this time, only capturing the energy consumed by the core's datapath. Developing transition-sensitive models for the control path would be extremely difficult. One way to model control path power would be analytically. In any case, for the *SimplePower* processor core, the energy consumed by the datapath is much larger than the energy consumed by the control logic due to the relatively simple control logic. The architecture simulated by *SimplePower* in this paper is the integer ISA of *SimpleScalar*, a five stage RISC pipeline. Functional unit energy models (for $2.0\mu, 0.8\mu$, and 0.35μ technology) have been developed for various units including flip-flops, adders, register files, multipliers, ALUs, barrel shifters, multiplexors, and decoders.

SimplePower outputs the energy consumed from one execution cycle to the next. It mines the transition sensitive energy models provided for each functional unit and sums them to estimate the energy consumed by each instruction cycle. The size of these energy tables could, however, become very large as the number of inputs to the bit-dependent functional units increase (for units like registers, each bit positions switching activity is independent and thus one small table characterizing one flip-flop is sufficient). To solve the table size problem, we partition the functional units into smaller sub-modules. For example, a register file is partitioned into five major sub-modules: five 5:32 decoders, word-line drivers, write data drivers, read sense-amplifiers, and a 32 × 32 cell array. Energy tables were constructed for each submodule. For example, a 1,024 entry table indexed by the pair of five current and five previous register select address bits was developed for the register file decoder component. This table is then shared by all the five decoders in the register file. Since the write data drivers, read sense amplifiers, word line dividers and all array cells are all bit-independent submodules, their energy tables are quite small. For the 32 × 32 5-port register file, our power estimation approach took much less than 0.1 seconds for each input transition as opposed to the 556.42 seconds required for circuit level simulation using HSPICE. The machine running the HSPICE simulation and our simulator is a Sun Ultra-10 with 640 MBytes memory. Our transition-sensitive modeling approach has been validated to be accurate (average error rate of 8.98%) using actual current measurements of a commercial DSP architecture [9].

As mentioned earlier, *SimplePower* currently uses a combination of *analytical and transition-sensitive energy models* for the memory system. The overall energy of the the memory system is given by

$$E_{memory} = E_{Icache} + E_{Dcache} + E_{Buses} + E_{Pads} + E_{MM}$$

The energy consumed by the instruction cache (Icache), E_{Icache}, and by the data cache (Dcache), E_{Dcache}, is evaluated using an analytical model that has been validated to be accurate (within 2.4% error) for conventional cache systems [13; 24]. We extended this model to consider the energy consumed during writes as well and have also parameterized the cache models to capture different architectural optimizations. E_{Buses} includes the energy consumed in the address and data buses between the Icache/Dcache and the datapath. It is evaluated by monitoring the switching activity on each of the bus lines assuming a capacitive load of 0.5pF per line. The energy consumed by the I/O pads and the external buses to the main memory from the caches, E_{Pads}, is evaluated similarly for a capacitive load of 20pF per line. The main memory energy, E_{MM}, is based on the model in [24] and assumes a per main memory access energy (refered as E_m in the rest of the paper) of 4.95×10^{-9}J based on the data for the Cypress CY7C1326-133 SRAM chip.

While the *SimplePower* framework models the influence of the clock on all components of the architecture (i.e., it assumes clock gating is implemented), it does not capture the energy consumed by the clock generation and clock distribution network. Existing clock energy estimation models [19; 8] require clock loading and physical dimensions of the design that can be obtained only after physical design and are difficult to estimate in the absence of structural information. However, we realize that this is an important additional component of the system energy consumption and we plan to address this in future research.

3. ENERGY DISTRIBUTION

With the emergence of energy consumption as a critical constraint in system design, it is essential to identify the energy hotspots of the system early in the design cycle. There has been significant work on estimating and optimizing the system power [7]. However, many have focused on estimating/optimizing only specific components of the system and most do not capture the integrated impact of circuit, architectural and software optimizations. Further, most existing high-level RTL energy estimation techniques provide a coarse grain of measurement resulting in 20-40% error relative to that of a transistor level estimator [2]. By contrast, *SimplePower* provides an integrated, cycle-accurate energy estimation mechanism that captures the energy consumed

Program	Source	# of Arrays	Input Size (KB)	Instruction Count	Dcache Miss Rates
tomcatv	Specfp95	9	119	4,868,048	0.182
nasa7/btrix	Specfp92	29	4,312	44,430,039	0.167
nasa7/mxm	Specfp92	3	120	47,168,707	0.112
nasa7/vpenta	Specfp92	9	114	2,274,945	0.143
adi	Livermore	6	241	6,062,860	0.098
dtdtz (aps)	Perfect Club	17	1,605	42,119,337	0.135
bmcm (wss)	Perfect Club	11	126	89,539,244	0.105
psmoo (tfs)	Perfect Club	1	204	16,955,980	0.077
eflux (tfs)	Perfect Club	5	297	12,856,306	0.114
amhmtm (wss)	Perfect Club	10	125	89,145,635	0.039

Table 1: Programs used in the experiments. Dcache miss rates are for 1K direct mapped caches with 32 byte line sizes. Instruction count is dynamic instruction count.

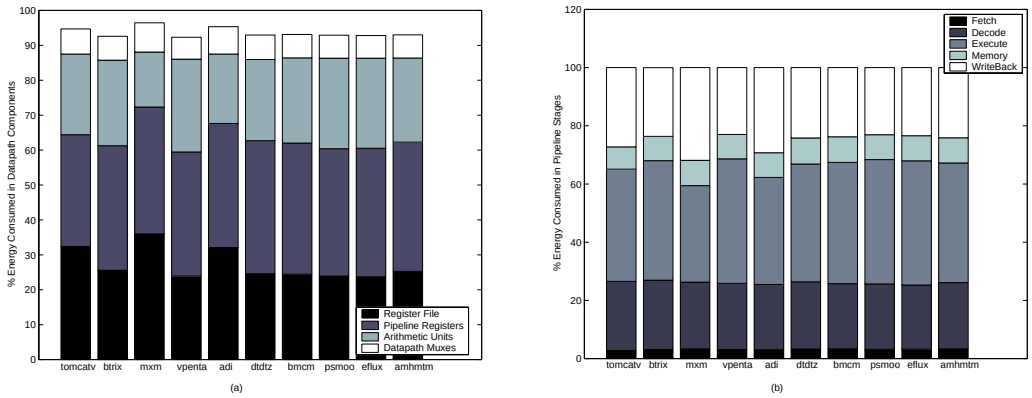

Figure 2: Energy distribution (%) of (a) the major energy consuming data path components and (b) the pipeline stages. The memory pipeline stage energy consumption does not include that of the ICache and DCache.

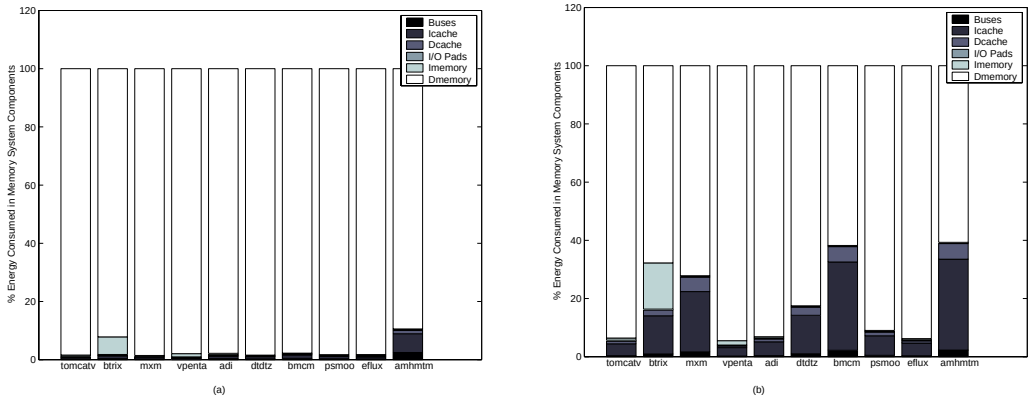

Figure 3: Energy distribution (%) in memory system: (a) 1K 4way Dcache and (b) 8K 4way Dcache. Imemory and Dmemory are the energies consumed in accessing the main memory for instructions and data, respectively.

	Original (Unoptimized)					Optimized				
	Datapath Energy (mJ)	Memory System Energy (mJ)				Datapath Energy (mJ)	Memory System Energy (mJ)			
		↓ →	1-way	2-way	4-way	8-way		↓ →	1-way	2-way	4-way	8-way
tomcatv	3.9	1K	357.2	223.3	200.1	179.7	4.2	1K	277.7	102.3	66.6	44.5
		2K	202.9	195.8	178.7	178.7		2K	69.6	64.4	34.4	34.5
		4K	75.8	171.1	172.7	175.4		4K	52.3	34.6	34.9	35.4
		8K	170.9	48.4	125.2	135.8		8K	45.5	35.9	36.9	37.6
		16K	140.5	133.1	61.9	39.1		16K	46.9	40.9	40.2	40.7
btrix	30.2	1K	1,788.8	1,162.8	939.4	942.5	28.8	1K	1,917.1	1,148.4	1,023.1	1,046.1
		2K	1,248.3	826.4	563.8	548.6		2K	1,287.9	965.0	675.6	581.2
		4K	1,023.2	513.6	432.6	371.5		4K	1,093.0	718.8	641.1	593.2
		8K	478.3	447.9	358.1	362.4		8K	737.1	690.8	604.2	604.8
		16K	446.5	345.6	348.3	353.0		16K	714.0	602.2	610.0	620.7
mxm	34.3	1K	1,718.1	1,502.8	1,490.8	495.2	83.7	1K	1,045.1	320.4	232.5	245.5
		2K	1,399.9	1,461.9	1,498.0	1,502.4		2K	627.2	208.4	153.4	163.0
		4K	1,123.9	522.7	405.3	267.5		4K	342.0	173.7	159.5	174.6
		8K	1,059.3	377.8	240.6	245.0		8K	300.5	192.6	196.4	199.6
		16K	1,070.1	379.3	297.3	302.0		16K	375.0	312.7	315.7	327.1
vpenta	1.6	1K	126.5	115.9	112.7	112.9	1.9	1K	149.3	101.0	77.5	71.3
		2K	120.0	112.9	113.0	113.2		2K	95.5	79.0	61.6	60.5
		4K	109.6	112.9	113.6	113.8		4K	77.2	60.6	58.9	59.0
		8K	78.4	80.9	82.7	87.4		8K	66.2	57.8	57.4	57.6
		16K	48.1	32.9	37.9	45.3		16K	59.6	57.2	57.1	57.3
adi	4.3	1K	240.5	154.6	137.8	136.0	5.2	1K	203.0	122.9	120.0	119.4
		2K	186.2	138.7	134.6	135.2		2K	144.9	111.0	117.0	120.5
		4K	166.9	136.7	136.4	136.9		4K	90.5	77.3	83.1	77.0
		8K	127.1	123.4	135.2	140.4		8K	65.5	51.5	50.4	51.0
		16K	114.9	87.3	97.6	101.4		16K	64.7	58.6	58.9	59.7
dtdtz	27.5	1K	2,149.3	1,402.8	1,146.1	1,064.8	31.2	1K	1,927.4	823.1	516.2	431.6
		2K	1,246.6	1,155.4	1,045.3	1,053.8		2K	610.3	459.8	306.8	326.4
		4K	880.5	857.7	815.1	861.2		4K	428.1	180.2	180.1	136.8
		8K	483.4	362.8	336.6	213.3		8K	326.9	176.1	158.9	163.3
		16K	341.5	235.9	191.6	195.3		16K	263.3	218.4	214.9	219.5
bmcm	59.2	1K	2,453.4	1,630.2	1,514.2	1,522.0	90.0	1K	2,399.0	1,055.9	844.7	804.9
		2K	1,566.9	1,493.8	1,526.9	1,534.9		2K	1,412.0	584.5	474.2	475.1
		4K	1,007.6	654.2	536.1	385.9		4K	724.4	416.8	289.8	227.5
		8K	467.6	365.5	312.1	318.9		8K	459.8	275.5	271.4	223.8
		16K	484.1	409.0	413.0	421.6		16K	488.6	347.2	351.8	364.6
psmoo	11.6	1K	519.5	432.4	426.0	427.6	16.1	1K	556.7	208.6	163.7	165.8
		2K	449.8	419.9	428.5	430.0		2K	180.8	173.1	164.7	168.3
		4K	341.9	340.7	328.9	343.9		4K	125.7	102.4	88.2	89.6
		8K	341.3	267.5	267.7	269.2		8K	91.8	81.1	81.5	84.1
		16K	352.7	281.3	281.7	283.4		16K	107.4	102.6	103.9	106.0
eflux	8.6	1K	463.8	377.3	372.9	374.1	10.1	1K	344.8	210.2	200.6	205.4
		2K	413.2	376.3	374.8	375.9		2K	255.8	196.4	192.7	190.6
		4K	383.0	364.9	368.6	379.6		4K	226.1	192.9	192.7	193.5
		8K	327.5	336.5	311.0	308.3		8K	215.1	200.3	200.6	201.9
		16K	309.6	258.2	264.2	263.9		16K	219.0	211.1	212.2	213.8
amhmtm	59.8	1K	1,070.4	348.7	275.9	275.2	66.5	1K	1,487.1	450.0	312.4	300.0
		2K	759.0	296.5	269.1	277.0		2K	998.9	327.0	264.3	273.1
		4K	623.7	271.1	259.9	265.1		4K	748.9	303.3	287.0	300.0
		8K	578.3	308.4	301.9	309.8		8K	551.3	217.5	232.7	265.3
		16K	447.2	403.7	403.2	411.8		16K	368.2	290.5	287.6	297.6

Table 2: Energy consumption for various Dcache configurations. For all the cases, an 8K direct-mapped Icache, 32 bytes line sizes, writeback policy and a core based on 0.8μ, 3.3V technology are used.

in the different components of the system.

In this section, we present the energy characteristics of ten benchmark codes written in the C language[2] (shown in Table 1) from the multidimensional array domain. An important characteristic of these codes is that they access large arrays using nested loops. The applications run on energy-constrained signal and video embedded processing systems exhibit similar characteristics. Since *SimplePower* currently works only with integer data types, floating point data accessed by these codes were converted to operate on integer data. In particular, memory access patterns (in terms of temporal and spatial locality) do not change. In order to limit the simulation times we scaled down the input sizes; however, all the benchmarks were run to completion. The experimental cache sizes (1K-16K) used in our study are relatively small as our focus is on resource-constrained embedded systems.

The energy consumed by the system is divided into two parts: *datapath energy* and *memory system energy*. The major energy consuming components of the datapath are the register file, pipelined registers, the functional units (e.g. ALU, multiplier, divider), and datapath multiplexers. The memory system energy includes the energy consumed by the Icache and Dcache, the address and data buses, the address and data pads and the off-chip main memory. Table 2 provides the energy consumption (in mJ) of our benchmarks for the datapath and memory system for various Dcache configurations. For all the cases in this paper, an 8K direct mapped Icache, line sizes of 32 bytes (for both Dcache and Icache), writeback cache policy, and a core based on 0.8μ, 3.3V technology were used. We also present only a single datapath energy value for the different configurations due to the efficient stall power reduction techniques (e.g., clock gating on the pipeline registers) employed in the datapath. With the aggressive clock gating assumed by *SimplePower*, the energy consumed during stall cycles was observed to be insignificant for our simulations. For example, `tomcatv` expends a maximum of 1% of the total datapath energy on stalls for all cache configurations studied.

We observe that the datapath energy consumption ranges from 1.577mJ to 59.776mJ for the various codes determined by the dynamic instruction length and the switching ac-

[2]Original codes are in Fortran and were converted into C by paying particular attention to the original data access patterns.

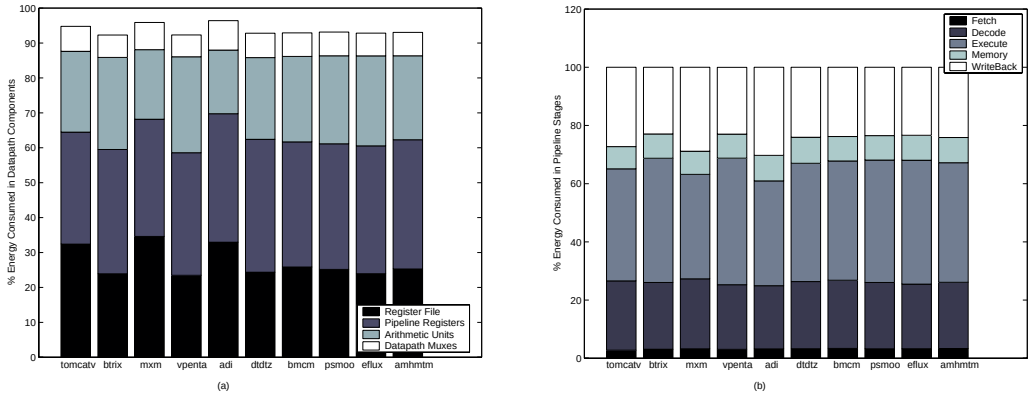

Figure 4: Energy distribution (%) of (a) the major energy consuming data path components and (b) the pipeline stages after applying code transformations. The memory pipeline stage energy consumption does not include that of the ICache and DCache.

tivity in the datapath. Compared to the memory system energy, the datapath energy is an order or two smaller in magnitude. This result corroborates the need for extensive research on optimizing the memory system power [25; 6; 13; 7]. Next, we zoom-in on the major energy consuming components of the datapath. It is observed from Figure 2(a) that the pipeline registers and register file form the energy hotspots in the datapath contributing 58-70% of the overall datapath energy. The extensive use of pipelining in DSP data paths to improve performance [1] and facilitate other circuit optimization such as voltage scaling will exacerbate the pipeline register energy consumption. Also, larger and multiple-port register files required to support multiple issue machines will increase the register file energy consumption further. The core energy distribution is also found to be relatively independent of the codes being analyzed. This is undoubtably impacted by only simulating integer data operations. The energy consumed by each stage of the pipeline is calculated by *SimplePower* and is shown in Figure 2(b). The decode stage energy does not include control logic energy consumption since it is not modeled by *SimplePower*. The pipeline register is the main contributer to the energy consumed in the memory stage, since ICache and DCache energy consumption is not included. The execution stage of the pipeline that contains the arithmetic units is the major energy consumer in the entire datapath, since the register file energy consumption is split between the decode and writeback stages.

The memory system energy consumption generally reduces with decrease in capacity and conflict misses when the DCache size or associativity is increased (see Table 2). Yet, in thirty seven out of the fifty cases, when we move from a 4way to 8way DCache, the memory system energy consumption increases. A similar trend is observed in fifteen out of forty cases when we move from an 8K to 16K Dcache. Moving to a larger cache size or higher associtivities increases the energy consumption per access. However, for many cases, this per access cost is amortized by the energy reduction due to a fewer number of accesses to the main memory. Of course, if the numbers of misses/hits are equal, using a less sophisticated cache leads to lower energy consumption. Figure 3(a) shows the energy distribution in the memory system components for a 1K 4way Dcache configuration where the main memory energy consumption dominates due to the large number of Dcache misses. For `btrix` and `amhmtm`, the data accesses per instruction are the smallest. In `amhmtm`, the majority of instruction accesses are satisfied from the Icache resulting in a more significant Icache energy consumption, whereas `btrix` exhibits a relatively poor instruction cache locality (the number of Icache misses is 100 times more than the next significant benchmark) resulting in increased energy consumption in main memory. When we increase the data cache size, the majority of data accesses are satified from the data cache. Hence, the overall contribution of the Icache and Dcache becomes more significant as observed from Figure 3(b).

SimplePower provides a comprehensive framework for identifying the energy hotspots in the system and helps the hardware and software designers focus on addressing these bottlenecks. The rest of this paper evaluates software and architectural optimizations targeted at addressing the energy hotspot of the system, namely, the energy consumed in data accesses.

4. IMPACT OF COMPILER OPTIMIZATIONS

To evaluate the impact of compiler optimizations on the overall energy consumption, we used a high-level compilation framework based on loop (iteration space) and data (array layout) transformations. For this study, the framework proposed in [14] was enhanced with iteration space tiling, loop fusion, loop distribution, loop unrolling, and scalar replacement. Thus, our compiler is able to apply a suitable combination of loop and data transformations for a given input code, with an optimization selection criteria similar to that presented in [14]. Our enhanced framework takes as input a code written in C and applies these optimizations (primarily) to improve temporal and spatial data locality. The tiling technique employed is similar to one explained in [27] and selects a suitable tile size for a given code, input size, and cache configuration. The loop unrolling algorithm carefully weighs the advantages of increasing register reuse and the disadvantages of larger loop nests in selecting an optimal degree of unrolling and is similar in spirit to the technique discussed in [5].

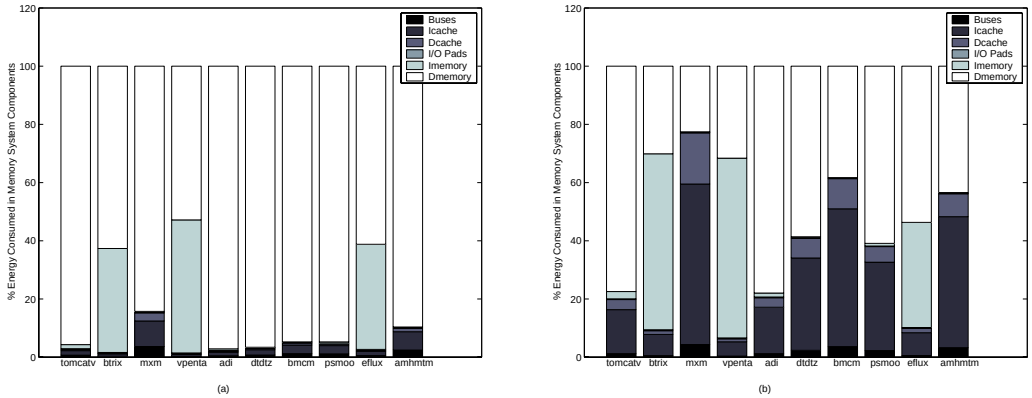

Figure 5: Energy distribution in memory system after applying code transformations: (a) 1K 4way Dcache and (b) 8K 4way Dcache. Imemory and Dmemory are the energies consumed in accessing the main memory for instructions and data, respectively.

There have been numerous studies showing the effectiveness of these optimizations on performance (e.g., [21; 28]); their impact on energy consumption of different parts of a computing system, however, remains largely unstudied. This study is important because these optimizations are becoming popular in embedded systems, keeping pace with the increased use of high-level languages and compilation techniques on these systems [18]. Through a detailed analysis of the energy variations brought about by these techniques, architects can see which components are energy hotspots and develop suitable architectural solutions to account for the influence of these optimizations.

Our expectation is that most compiler optimizations (in particular when they are targeted at improving data locality) will reduce the overall energy consumed in memory subsystem. This is a side effect of reducing the number of off-chip data accesses and satisfying the majority of the references from the cache. Their impact on the energy consumed in the datapath, on the other hand, is not as clear. As observed in Section 3, the energy consumed in the memory subsystem is much higher than that consumed in the datapath. While this might be true for unoptimized codes (due to the large number of off-chip accesses), it would be interesting to see whether this still holds after the locality-enhancing compiler optimizations.

Table 2 also shows the resulting datapath and memory system energy consumption as a result of applying our compiler transformations. The most interesting observation is that the optimizations increase the datapath power for all codes except `btrix`. This increase is due to more complex loop structures and array subscript expressions as a result of the optimizations. Since, in optimizing `btrix`, the compiler used only *linear* loop transformations (i.e., the transformations that contain only loop permutation, loop reversal, and loop skewing [28]), the datapath energy did not increase. Next, we observe that the reduction in the memory system energy makes the datapath power more significant. For example, after the optimizations, in the `mxm` benchmark, the datapath power constitutes 29% of the overall system energy for a 8K 8way cache configuration (as compared to 12.3% before the optimizations). In fact, the datapath power becomes larger than that consumed in the memory system if we do not consider the energy expended in instruction accesses. This is significant as our optimizations were targeted only at improving the data cache performance. Thus, it is important for architects to continue to look at optimizing the datapath energy consumption rather than focus only on memory system optimizations.

The compiler optimizations had little effect on the energy distribution on the datapath components and pipeline stages as shown in Figures 4(a) and (b). However, the energy distribution (shown in Figure 5) in the memory system shows distinct differences from the unoptimized (original) versions (see Figure 3). In the optimized case, the relative contribution of the main memory is significantly reduced due to more data cache hits. Hence, we observe that the contribution of the Icache and Dcache energy consumption becomes more significant for all optimized codes that we used. Thus, energy-efficient Icache and Dcache architectures become more important when executing the compiler optimized codes. The effectiveness of architectural and circuit techniques to design energy-efficient caches is discussed in Section 5.

As mentioned earlier, normally, our compiler automatically selects a suitable set of optimizations for a given code and cache topology. Since, in doing so, it uses heuristics, there is no guarantee that it will arrive at an optimal solution. In addition to this automatic optimization selection, we have also implemented a directive-based optimization scheme which relies on user-provided directives and, depending on them, applies the necessary loop and data transformations. Next, we forced the compiler using these compiler directives to apply all eight combinations of three mainstream loop optimizations [28], namely, loop unrolling, tiling, and linear loop transformations to the `mxm` benchmark. The results presented in Figure 6 reveal that the best compiler transformation from the energy perspective varies based on the cache configuration. This observation presents a new challenge for the compiler writers of embedded systems, as the most aggressive optimizations (although they may lead to minimum execution times) do not necessarily result in the best code from the energy point of view.

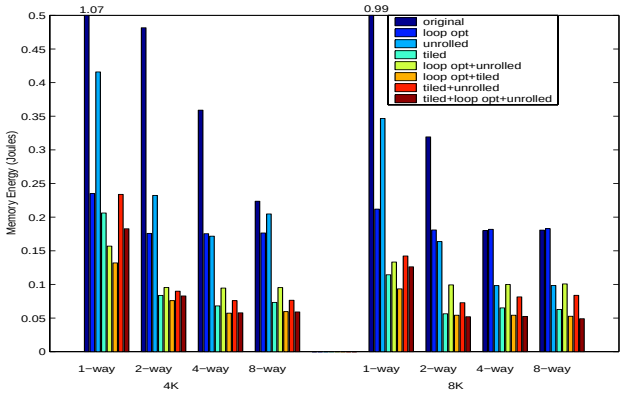

Figure 6: Energy distribution in the memory system (with different DCache configurations) as a result of different code transformations. `original` is the unoptimized program, `loop opt` denotes the code optimized using linear loop transformations, `unrolled` denotes the version where loop unrolling is used and `tiled` is the version when tiling is applied.

5. ENERGY EFFICIENT CACHE ARCHITECTURES

The study of cache energy consumption is relatively new and the optimization techniques can be broadly classified as circuit and architectural. The main circuit optimizations include activating only a portion of the cells on the bit (DBL) and word lines, reducing the bit line swings using pulsed word lines (PWL) and isolated sense amplifiers (IBL), and charge recycling in the I/O buffer [12]. The application of these optimizations is independent of the code sequences themselves. Many architectural techniques have been proposed as optimizations for the memory system [25; 13; 16]. Many of these techniques introduce a new level of memory hierarchy between the cache and the processor datapath. For instance, the work by Kin et. al. [16] proposed accessing a small filter cache before accessing the first level cache. The idea is to reduce the energy consumption by avoiding access to a larger cache. While such a technique can have a negative impact on performance, it can result in significant energy savings. The block-buffering (BB) mechanism [13] uses a similar idea by accessing the last accessed, buffered cache line before accessing the cache. Unlike circuit optimizations, the effectiveness of these architectural techniques is influenced by the application characteristics and the compiler optimizations used. For instance, software techniques can be used to improve the locality in a cache line by grouping successively accessed data. Then, a cache buffering scheme can exploit this improved locality. Thus, increasing spatial locality within a cache line through software techniques can save more energy. A detailed study of such interactions between software optimizations and the effectiveness of energy-efficient cache architectures will be useful to both compiler writers and hardware designers.

To capture the impact of circuit optimization in the energy estimation framework, we measured the influence of applying different combinations of circuit optimizations using four different layouts of a 0.5Kbits SRAM using HSPICE simulations. It was observed that the energy consumed can be reduced on an average by 29% and 52% as compared to an unoptimized SRAM when applying the (PWL+IBL) and (PWL+IBL+DBL) optimizations. We conservatively utilize the 29% reduction achieved by the (PWL+IBL) scheme to capture the efficiency of the circuit optimizations in our analytical model for memory system energy. We refer to the (PWL+IBL) scheme as IBL in the rest of this paper for convenience.

First, we studied the interaction between the compiler optimizations and the effectiveness of the BB mechanism. In order to study this interaction, the Dcache was enhanced to include a buffer for the last accessed set of cache blocks (one block buffer for each way). A code that exhibits increased spatial and temporal locality can effectively exploit the buffer. We define the *relative energy savings ratio* of an optimized code (*opt*) over an unoptimized code (*orig*) for a given hardware optimization *hopt* as:

$$Relative\ energy\ savings\ ratio_{hopt} =$$

$$100 * \frac{[(E\_optcode - E\_optcode_{hopt}) - (E\_orig - E\_orig_{hopt})]}{(E\_orig - E\_orig_{hopt})}$$

where $E\_optcode$, E_orig are the energy consumed due to the execution of optimized and unoptimized code respectively without *hopt*, and $E\_optcode_{hopt}$, E_orig_{hopt} are the corresponding values with *hopt*. This measure enables us to evaluate the effectiveness of compiler optimizations in exploiting the hardware optimization technique. Figure 7 shows the relative energy savings ratio for BB. It can be observed that the block buffer mechanism was more effective in reducing energy for the optimized codes (except for `eflux`). This is due to the better spatial and temporal locality exhibited by the compiler optimized codes. This improved locality results in more hits in the block buffer. On an average, the optimized codes achieve 19% (18%) more energy savings relative to the original codes using a direct-mapped (4way) cache with BB. The reason that optimized `eflux` code does not take better advantage of BB than unoptimized code is that the accesses with temporal locality in the unoptimized code were better clustered, leading to increased data reuse in the block buffer. Next, we applied a combination of the IBL and BB and executed the *optimized* codes to find the combined effect of circuit, architectural and software optimizations on the overall memory system energy. It can be observed from Figure 8 that the Dcache energy consumption can be reduced by 58.8% (58.7%) for the direct-mapped (4way) cache configuration. Thus, architectural and circuit techniques working together can reduce the energy consumption of even highly optimized codes significantly. While the BB and IBL optimizations are very effective for reducing the energy consumed in the Dcache, it is important to investigate their impact on the overall memory energy reduction. It was found that the memory system energy reduces by 6.7% (11%) using the direct-mapped (4way) cache configuration (see Figure 9).

We also investigated the influence of the BB+IBL optimization for the Dcache due to the reduction in the energy per main memory access (E_m) as a result of emerging technologies such as the embedded DRAM (eDRAM) [22]. Figure 10 shows that the combined BB and IBL technique reduces memory system energy by 27.7% with new (future) tech-

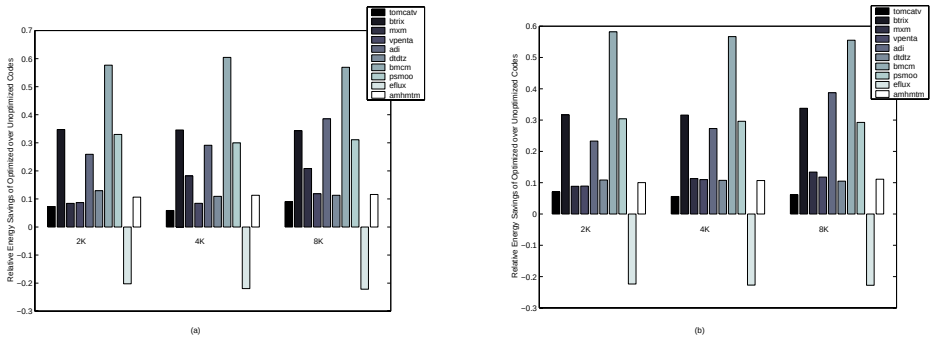

Figure 7: Relative energy savings ratio of Dcache for optimized code over unoptimized code using BB on (a) 1way Dcaches and (b) 4way Dcaches.

nologies that have a potential to reduce the per access energy by an order of magnitude (We use a E_m=4.95e-10J) as compared to the 16.3% reduction in current technology (E_m=4.95e-9J). *SimplePower* can similarly be used to evaluate the influence of other new technologies and energy-efficient techniques such as BB on the energy consumed by the system as a whole and an individual component in particular.

Next, we evaluated the combination of a most recently used way-prediction cache and BB mechanism. The way-prediction caches have been used to address the longer cycle time in associative caches as compared to direct mapped caches [4]. While most prior effort has focussed on way-prediction caches for addressing the performance problem, the energy efficiency of these cache architectures was evaluated recently by Inoue et. al. [10]. In their work, an MRU (Most Recently Used) algorithm that predicts and probes only a single way first was used. If the prediction turns out to fail, all remaining ways are accessed at the same time in the next cycle. We refer to this technique as the MRU scheme and the caches that use them as MRU caches. It must be noted that MRU caches could increase the cache access cycle time [4; 15]. However, our work focus is on energy estimation and optimization rather than investigating energy-performance tradeoffs. Here, we study the effectiveness of combining two different architectural techniques to optimize system energy and also evaluate the impact of software optimizations enabled by the *SimplePower* optimizing compiler on the MRU prediction.

We studied the energy savings that can be obtained using MRU caches for 4way associative cache configurations. It can be observed from Figure 11 that the optimized codes benefit more from the MRU scheme and can obtain 21% more savings than the original code on an average. The increased locality in the optimized codes increases the number of successful probes in the predicted way of the MRU cache. We also find that using the MRU scheme reduces Dcache energy by 70.2% on an average for optimized codes as compared to using a conventional 8K, 4way associative caches (see Figure 12(a)). The incremental addition of BB and IBL provided additional 10.5% and 5.5% energy reduction respectively. Figure 12(b) shows the energy savings in the entire memory system are 23%, 24.2% and 26.4% when MRU, BB and IBL are applied incrementally in that order.

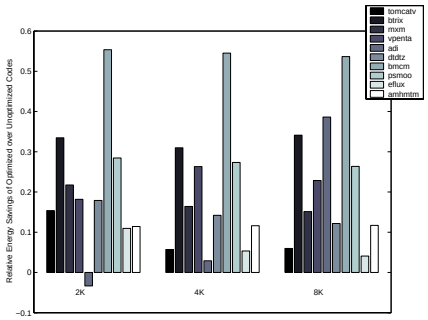

Figure 11: Relative energy savings ratio of Dcache for optimized code over unoptimized code using MRU for 4way Dcaches.

From the study in this section, we find that the optimized codes are not only efficient in reducing the number of costly (in terms of energy) accesses to main memory but they are also more effective in exploiting the energy efficient architectural mechanisms such as MRU caches and BB. We also find that the incremental benefits of applying the BB scheme over a MRU cache is significantly smaller as compared to using these techniques individually. A designer can use similar early energy estimates provided by *SimplePower* to perform energy-cost-performance tradeoffs for new energy efficient techniques.

6. IMPLICATIONS OF ENERGY-EFFICIENT MEMORY SYSTEMS

Emerging new technologies combined with the energy-efficient circuit, architectural and compiler techniques for reducing memory system energy can potentially create a paradigm shift in the importance of energy optimizations from the memory system to the datapath and other units. Here, we consider the influence of changes in the energy consumed per main memory access, E_m. Such changes are eminent due to new process technologies [22] and reduction in physical distance between the main memory and the datapath. Table 3 shows the memory system energy for different values of E_m for four different cache organizations using two optimized codes. Note that $E_m = 4.95 \times 10^{-9}$J is the value that we have used so far in this paper. The lowest E_m value that we experiment with in this section (4.95×10^{-11}) corresponds to

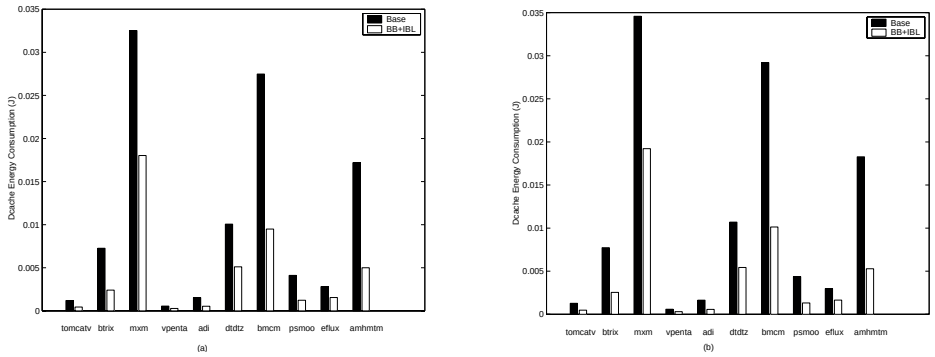

Figure 8: Dcache energy consumption of optimized codes using (BB + IBL) for (a) 1way 8K Dcaches and (b) 4way 8K Dcaches.

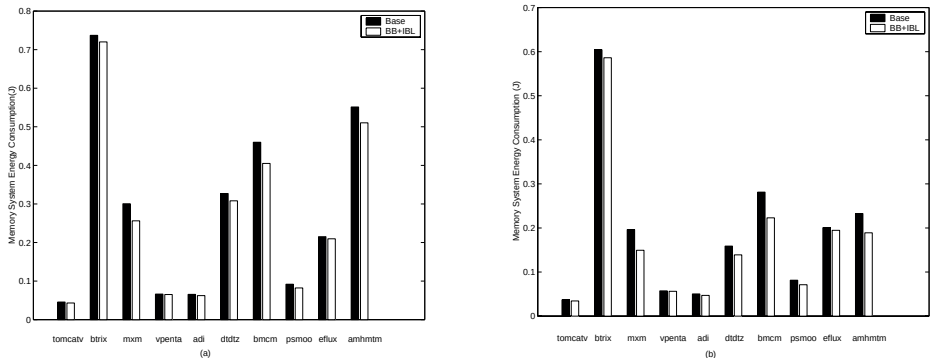

Figure 9: Memory system energy for optimized codes using (BB + IBL) using (a) 1way 8K Dcaches and (b) 4way 8K Dcaches.

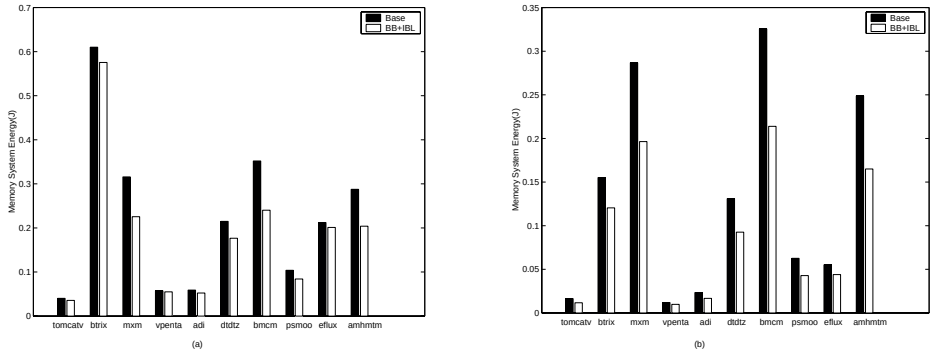

Figure 10: Memory system energy for optimized codes using (BB + IBL) using 4way 16K Dcaches with (a) $E_m = 4.95e-9$J (b) $E_m = 4.95e-10$J

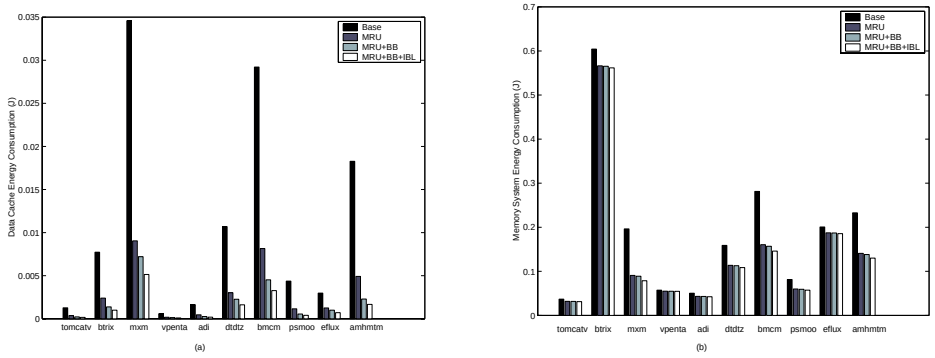

Figure 12: Energy consumption when a combination of MRU, BB and IBL techniques are applied to an 8K, 4way associative cache configuration (Base) in (a) Dcache (b) Memory System.

the magnitude of energy per first-level on-chip cache access with current technology.

Recall that the datapath energy consumption for the optimized `mxm` and `psmoo` codes were 83.7mJ and 16.1mJ, respectively (see Table 2). Considering the fact that large amounts of main memory storage capacity are coming closer to the CPU [22], we expect to see E_m values lower than $E_m = 4.95 \times 10^{-9}$J in the future. Such a change could make the energy consumed in the datapath larger than the energy consumed in memory. For example, with $E_m = 4.95 \times 10^{-10}$ and a 1K, 4-way cache, the energy values of datapath becomes larger than that of the memory system for `mxm`.

7. CONCLUSIONS

The need for energy efficient architectures has become more critical than ever with the proliferation of embedded devices. Also, the increasing complexity of the emerging systems on a chip paradigm makes it essential to make good energy-conscious decisions early in the design cycle to help define design parameters and eliminate incorrect design paths. This study has introduced a comprehensive framework that can provide such early energy estimates at the architectural level. The uniqueness of this framework is that it captures the integrated impact of both hardware and software optimizations and provides the ability to study the system as a whole and each individual component in isolation. This work has tried to answer some of the questions raised in Section 1 using this framework. The major findings of our research are the following:

- A transition-sensitive, cycle-accurate, architectural-level approach can be used to provide a fast (as compared to circuit-level simulators) and relatively accurate estimate of the energy consumption of the datapath. For example, the register file energy estimates from our simulator are within 2% of circuit level simulation.

- The energy hotspots in the datapath were identified to be the pipeline registers and the register file. They consume 58-70% of the overall datapath energy for executing (original) unoptimized codes. However, the datapath energy is found to be an order or two magnitude less the memory system energy for these multidimensional array codes.

- The main memory energy consumption accounts for almost all the system energy for small cache configurations when executing unoptimized codes. The application of high-level compiler optimizations significantly reduces the main memory energy causing the Dcache, Icache and datapath energy contributions to become more significant. For example, the contribution of datapath energy to overall system energy, with an 8K, 8way Dcache, increases from 12.3% to 29.5% when benchmark `mxm` is optimized.

- The improved spatial and temporal locality of the optimized codes is useful in not only reducing the accesses to the main memory but also in exploiting energy-efficient cache architectures better than with unoptimized codes. Optimized codes saved 21% times more energy using the most recently used way-predicting cache scheme as compared to executing unoptimized codes. They also save 19% more energy when using block buffering.

- Emerging technologies coupled with a combination of energy-efficient circuit, architectural and compiler optimizations can shift the energy hotspot. We found that with an order of magnitude reduction in main memory energy access made possible with eDRAM technology, the datapath energy consumption becomes larger than the memory system energy when executing an optimized `mxm` code with a 1K 4way Dcache.

In this work, we observed that the compiler optimizations provided the most significant energy savings over the entire system. The *SimplePower* framework can also be used for evaluating the effect of high-level algorithmic, architectural, and compilation trade-offs on energy. Also, we observed that energy-efficient architectures can reduce the energy consumed by even highly optimized code significantly and, in fact, much better than with unoptimized codes. An understanding of the interaction of hardware and software optimizations on system energy gained from this work can help both architects and compiler writers to develop more energy-efficient systems.

This paper has looked at only a small subset of issues with respect to studying the integrated impact of hardware-software optimizations on energy. There are a lot of issues that are ripe for future research. The interaction of algorithmic selec-

mxm

Configuration	Memory Energy (mJ)							
	4.95×10^{-11}	2.475×10^{-10}	4.95×10^{-10}	2.475×10^{-9}	4.95×10^{-9}	2.475×10^{-8}	4.95×10^{-8}	2.475×10^{-7}
1K, 1way	41.3	81.9	132.6	538.2	1,045.2	5,101.1	10,171.2	50,731.1
1K, 4way	38.1	45.9	55.8	134.3	232.6	1,018.4	2,000.9	9,860.0
4K, 1way	80.8	91.3	104.5	210.1	342.0	1,397.9	2,717.6	13,275.3
4K, 4way	86.2	89.2	92.9	122.5	159.6	455.9	826.2	3,788.9

psmoo

Configuration	Memory Energy (mJ)							
	4.95×10^{-11}	2.475×10^{-10}	4.95×10^{-10}	2.475×10^{-9}	4.95×10^{-9}	2.475×10^{-8}	4.95×10^{-8}	2.475×10^{-7}
1K, 1way	13.1	35.0	62.5	282.2	556.8	2,753.4	5,499.3	27,466.1
1K, 4way	9.4	15.6	23.4	85.8	163.7	787.3	1,566.8	7,802.6
4K, 1way	17.3	21.7	27.2	70.9	125.8	563.9	1,111.7	5,493.6
4K, 4way	18.4	21.3	24.8	52.9	88.2	370.2	723.0	3,542.1

Table 3: Impact of different E_m values on total memory system energy consumption for optimized `mxm` and `psmoo`.

tion, low-level compiler optimizations and other low-power memory structures will be addressed in our future work.

8. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers whose commments helped to improve this paper. This work was sponsored in part by grants from NSF (MIP-9705128), Sun Microsystems, and Intel.

9. REFERENCES

[1] B. Ackland and C. Nicol. High performance DSPs - what's hot and what's not? In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 1–6, August 1998.

[2] D. Blaauw, A. Dharchoudhury, R. Panda, S. Sirichottiyakul, C. Oh, and T. Edwards. Emerging power management tools for processor design. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 143–148, August 1998.

[3] D. Burger and T. Austin. The simplescalar tool set, version 2.0. Technical report, Computer Sciences Department, University of Wisconsin, June, 1997.

[4] B. Calder, D. Grunwald, and J. Emer. Predictive sequential associative cache. In *Proceedings of Second International Symposium on High-Performance Computer Architecture*, pages 244–253, 1996.

[5] S. Carr and Y. Guan. Unroll-and-jam using uniformly generated sets. In *Proceedings of the 30th International Symposium on Microarchitecture*, Research Triangle Park, NC, December 1997.

[6] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom memory management methodology – exploration of memory organization for embedded multimedia system design*. Kluwer Academic Publishers, June 1998.

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

[8] R. Chen, N. Vijaykrishnan, and M. J. Irwin. Clock power issues in system-on-chip designs. In *Proceedings of the Annual IEEE CS Workshop on VLSI*, pages 48–53, 1999.

[9] R. Y. Chen, R. M. Owens, and M. J. Irwin. Validation of an architectural level power analysis technique. In *Proceedings of the Design Automation Conference*, page 242, June 1998.

[10] K. Inoue, T.Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 273–275, 1999.

[11] M. J. Irwin and N. Vijaykrishnan. Energy issues in multimedia systems. In *Proceedings of Workshop on Signal Processing Systems*, pages 24–33, October 1999.

[12] K. Itoh, K. Sasaki, and Y. Nakagome. Trends in low-power ram circuit technologies. *Proceedings of IEEE*, pages 524 –543, Vol. 83. No. 4, April 1995.

[13] M. Kamble and K. Ghose. Analytical energy dissipation models for low power caches. In *Proceedings of International Symposium on Low Power Electronics and Design*, page 143, August 1997.

[14] M. Kandemir, A. Choudhary, J. Ramanujam, and P. Banerjee. Improving locality using loop and data transformations in an integrated framework. In *Proceedings of the 31st International Symposium on Microarchitecture*, pages 285–296, December, 1998.

[15] R. R. Kessler, R. Jooss, A. Lebeck, and M. D. Hill. Inexpensive implementations of self-associativity. In *16th Annual International Symposium on Computer Architecture*, 1989.

[16] J. Kin, G. M. Gupta, and W. H. Mangione-Smith. The filter cache : an energy efficient memory structure. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 184–193, 1997.

[17] Y. Li and J. Henkel. A framework for estimating and minimizing energy dissipation of embedded hw/sw systems. In *Proceedings of the Design Automation Conference*, pages 188–191, 1998.

[18] S. Y. Liao. *Code Generation and Optimization for Embedded Digital Signal Processors*. PhD thesis, Dept. of EECS, MIT, Cambridge, Massachusetts, June 1996.

[19] D. Liu and C. Svensson. Power consumption estimation in cmos vlsi chips. *IEEE Journal of Solid-State Circuits*, page 663, June 1994.

[20] H. Mehta, R. M. . Owens, and M. J. Irwin. Energy characterization based on clustering. In *Proceedings of the Design Automation Conference*, page 702, June 1996.

[21] S. S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufmann Publishers, San Francisco, California, 1997.

[22] Y. Nunomura and et.al. M32R/D–Integrating DRAM and microprocessor. In *IEEE MICRO*, November/December 1997.

[23] K. Roy and M. C. Johnson. Software design for low power. *NATO Advanced Study Institute on Low Power Design in Deep Sub-micron Electronics*, August 1996.

[24] W.-T. Shiue and C. Chakrabarti. Memory exploration for low power, embedded systems, CLPE-TR-9-1999-20. Technical report, Arizona State University, 1999.

[25] C. L. Su and A. M. Despain. Cache designs for energy efficiency. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, January 1995.

[26] V. Tiwari, S. Malik, A. Wolfe, and T. Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing Systems*, Vol. 13, No. 2, August 1996.

[27] M. Wolf, D. Maydan, and D. Chen. Combining loop transformations considering caches and scheduling. In *Proceedings of the Annual International Symposium on Microarchitecture*, page 274, December 1996.

[28] M. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley Publishing Company, 1996.