Energy Minimization of a Pipelined Processor using a Low Voltage Pipelined Cache

Jun Cheol Park, Vincent J. Mooney III, Krishna Palem and Kyu-won Choi
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332
{jcpark, mooney, palem, kwchoi}@ece.gatech.edu

Abstract

A cache is a power-hungry component in a processor. Therefore, a reduction in cache energy can have a significant impact on overall processor energy consumption. In this paper, we propose a new energy minimization technique for a pipelined processor using a low voltage pipelined cache. We consider a case where a pipelined cache is not required but is used nonetheless, enabling the cache supply voltage to be lowered. Using this method, we show five benchmarks where, on average, power consumption is reduced by 24.85% at a cost of an average increase in execution time of 15.35% resulting in an average overall energy reduction of 13.33%.

1 Introduction

Power and energy reduction is not an optional condition in the embedded systems area. Especially for mobile devices, power and energy usage can be the most important issue due to battery limitations. Even general purpose high-performance processors require more attention to reducing power consumption.

Many ideas have been proposed to reduce power and/or energy without loss of performance. In this paper, however, we trade off performance (execution time) for power by introducing a technique that can reduce power and/or energy by integrating two different techniques used previously for different purposes. The first technique is a dual voltage system. A dual voltage system supplies two different voltages to different circuits in a chip. The lower voltage is used for non-critical path circuits to reduce power. However, the lower voltage typically cannot be used for circuits on the critical path, even though the critical path circuitry is a power-hungry component. One way to tackle this problem (the problem of not being able to use a lower voltage for a component on the critical path) is by pipelining the component. Then the critical path circuitry is split among multiple stages to that a lower supply voltage can be used.

We targeted L1 caches as power-hungry components on the critical path in a pipeline stage. We divided both instruction and data L1 caches into two stages and pipelined them with the corresponding processor stages and applied a lower supply voltage to the caches

The rest of the paper is organized as follows: Section 2 discusses the motivation for this work and overview of previous work. Section 3 describes the proposed pipelined cache architecture. Section 4 discusses the methodology. Section 5 presents the results and Section 6 concludes the paper.

2 Motivation and Previous Work

A cache is a local temporary storage of memory used to reduce time spent accessing memory. Every modern microprocessor has one or more levels of on-chip cache(s) to maximize performance. However, a cache is a power-hungry component. For example, Brooks et al. claim that instruction and data caches consume 22.2% and 11.1%, respectively, of total power in the Intel Pentium Pro [1]. Also, Montanaro et al. report an instruction cache and a data cache of a StrongARM 110 processor that consume 43% of the total chip power [2]. Therefore, caches are an important target for power reduction, and various approaches have been proposed. One good approach to reduce power is voltage scaling. Voltage scaling applies a different voltage (and frequency) to reduce power at a cost of slower circuitry. However, due to the increase of execution time, the total energy $(power \times execution\ time)$ may not change. The Intel XScale processor supports a range of frequencies and voltages in order to allow the user to save power [3]. Instead of scaling the voltage supply of the entire processor, Moshnyaga and Suji applied dual voltage supplies for a block-buffered cache [4]. The proposed cache uses a high voltage in case of a block miss that takes longer time, but uses a lower voltage for a block hit thus saving power. The main idea of this approach is to use a high voltage supply for the critical path and a low voltage supply for the non-critical path of the cache. The advantage of this technique is that the clock frequency need not be changed, so there is no performance loss. In other previous work, Igarashi proposed a method to find the critical path in a circuit and apply a lower supply voltage to non-critical paths. Instead of partitioning a circuit into two sub-circuit groups to sup-

ply different voltages, Igarashi introduced secondary lower voltage Vdd wires into the circuit. This can avoid long interconnections between the original voltage circuit and the lower voltage circuit, a problem of the conventional layout method, but results in significant area overhead [5].

Pipelined caches were originally proposed to improve performance. Chappel broke down a cache into two or more segments and placed latches between each stage to pipeline the cache [6]. Pipelining yields a reduction in average execution time per instruction [7]. Some processors, such as Intel XScale, implement a pipelined cache for performance enhancement [3].

In this paper, we propose a new way to reduce a cache energy by pipelining the cache and applying different voltages for the processor core versus the cache.

3 Approach

The cycle time of a processor is decided by the maximum delay critical path among the pipeline stages. We consider the case where a processor has been designed with a small number of pipeline stages (say, five) at a low clock frequency (say, 100 MHz) for low power. Given such a design, we consider the following options: (i) non-pipelined (single cycle access) caches of the same voltage as the processor and (ii) caches pipelined into two stages and supplied with a lower $V\,dd$.

In the described scenarios, the cycle time is already fixed (e.g. at 10 ns) by the processor pipeline. This, for case (ii) pipelined caches, each cache pipeline stage delay is already set by the worst case processor pipeline stage delay. Thus, power for case (ii) is minimized by lowering the cache supply voltage until the worst case cache delay of any pipeline stage equals the worst case processor pipeline stage delay. For a processor designed to run at 100 MHz, this worst-case delay is 10 ns. Obviously, reducing the cache supply voltage reduces the energy consumed by the cache.

But pipelining a cache does not come only with advantages. There are two disadvantages to consider in terms of data path pipeline performance. One is branch penalty, and the other is load use penalty. The branch penalty must be considered with a pipelined instruction cache. All modern processors use some kind of branch prediction scheme to hide branch penalty. If the branch target is mispredicted, some instructions are needlessly fetched and thus must be wiped out. The number of penalty cycles differs according to the pipeline architecture. In the case of a pipelined instruction cache, one or more cycles of penalty are added to the branch prediction penalty of an equivalent non-pipelined cache.

When pipelining the data cache is applied, the load use delay can be a problem. The load use delay occurs when a load instruction is immediately followed by a dependent instruction. A pipelined data cache requires one or more stall cycles according to the depth of the data cache pipeline. Various hardware and software techniques can be used to hide these penalties. Olukotun, Mudge and Brown explored various depths of a pipelined cache from a performance perspective [8].

4 Methodology

4.1 Processor Model

For our processor model we use MARS – obtained from the University of Michigan – a cycle-accurate Verilog Model of a five-stage RISC processor capable of running ARM instructions [9]. The five stages are instruction fetch (IF), instruction decode (ID), execution (EX), memory access (ME) and write back (WB). MARS has a non-pipelined instruction cache and a non-pipelined data cache. MARS uses backward-taken forward not-taken (BTFN) branch prediction scheme to hide branch penalties. We use MARS as is for our case (i) described in Section 3: processor with non-pipelined data and instruction caches.

We will describe this in more detail later in this section, but, in summary, we simulate MARS in the Synopsys VCS simulator and the Synopsys Power Compiler [10] in order to obtain both processor execution time measurements and processor core power measurements.

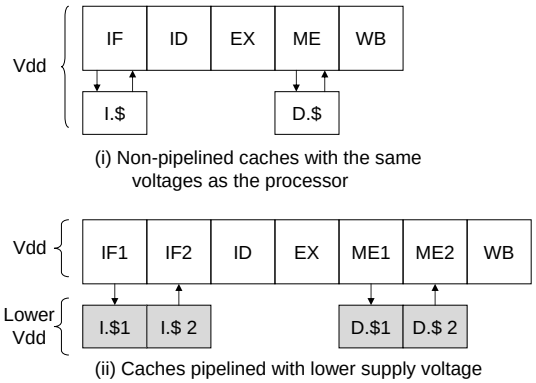

Figure 1: Proposed approach

We modified MARS to obtain case (ii) from Section 3: processor with two-stage pipelined data and instruction caches supplied with a lower voltage than the processor core. To alter MARS to use pipelined caches, one stage should be inserted between IF and ID for the pipelined instruction cache, and one another stage should be inserted between ME and WB for the pipelined data cache. We found that the BTFN branch prediction scheme is not adequate for the pipelined instruction cache because this prediction scheme predicts the next instruction in the ID stage. Instructions are decoded in the ID stage. Therefore, a processor recognizes a branch instruction at the end of the ID stage. As a result, the BTFN can hide some of the branch penalty, but BTFN still has a non-zero branch penalty even when the prediction is correct. Furthermore, if the instruc-

tion cache is pipelined resulting in two IF stages prior to the ID stage, then the branch penalty for a correct prediction increases by one cycle. A Branch Target Buffer (BTB), on the other hand, predicts the next instruction during the IF stage. Therefore, we can potentially have a branch penalty of zero, if the BTB predicts correctly [7].

We modified MARS so that MARS can simulate a pipelined cache. Figure 1 shows the result of modifying MARS to handle pipelined caches: we transformed the original 5-stage design (case(i)) into a 7-stage processor pipeline (case(ii)). For the pipelined instruction cache, we added an IF2 stage using a latch between the IF1 stage and ID stage. The IF1 stage was the IF stage in case (i). Next, we modified the branch control logic so that it calculates the target program counter (PC) value adjusted by the newly added IF2 stage. Second, we added an ME1 stage also using a latch between the EX stage and the ME2 stage for pipelined data cache simulation. The ME2 stage was the ME stage in case (i). Furthermore, we added one more data forwarding path from the ME1 stage to EX stage on top of the existing data forwarding paths of EX stage to EX stage and ME2 to EX stage. We also changed the branch prediction scheme. To predict the next instruction in the IF stage, we added a 128-entry branch target buffer (BTB) that has branch instruction address and branch target address. The BTB also has a 2-bit counter for branch prediction [11]. Once the BTB mispredicts, it updates BTB table while the pipeline wipes out the instructions inside.

Our simulation procedures are as follows. We use the GNU-gcc ARM cross compiler version egcs-2.91.66. For each benchmark we consider, we compile the benchmark to relocatable ARM assembly code using GNU-gcc ARM cross compiler. Then we use the GNU cross-assembler to generate a binary executable targeted towards ARM architectures. Next we translate the binary into an ASCII format called VHX (Verilog Hex) that is suitable for being simulated on MARS using Synopsys VCS [10]. We simulated the base architecture and the pipelined cache architecture with each of the five benchmarks. For each benchmark (with specific input data) and architecture, switching activity for the processor core and cache statistics are collected. The cache access statistics are fed to the CACTI memory models.

We use a synthesis based methodology for developing the power models for the submodules belonging to the data path (we consider the data path to consist of the fetch unit, decode unit, register file, arithmetic logic unit, data cache access unit and write-back unit). The synthesis infrastructure consists of two software tools from Synopsys, Inc.: the Design Compiler and Power Compiler [10].

Design Compiler generates the gate level netlist from the hardware description of the submodules given as Verilog RTL description. The netlist is generated using the TSMC 0.25μ library from LEDA systems [12]. The technology details include features like transistor width, transistor length, each gate capacitance, drain capacitance, rise time and fall time of each transistor. The synthesis process is guided by fixing the maximum delay and maximum area. The maximum delay was set to 10 ns and the maximum area was fixed to infinity so as to obtain the fastest implementation. In our case, the modules synthesized to operate at 100 MHz (i.e., a 10 ns cycle time).

The Power Compiler from Synopsys is used to estimate the power of the processor core. Switching activities of five benchmarks from VCS simulation is fed to the Power Compiler as an input file. Then, the Power Compiler reports dynamic and static power dissipation of the technology chosen. More details about the infrastructure can be obtained from a technical report [13].

4.2 Cache Timing and Power Model

Description	Parameters
Cache size	16KB, 32KB, 64KB, 128KB, 256KB, 512KB
Block size	32bytes
Associativity	directed mapped
Number of sets	512, 1K, 2K, 4K, 8K, 16K
Number of segments per word line (data)	1
Number of segments per bit line (data)	1
Number of segments per word line (tag)	1
Number of segments per bit line (tag)	1
Number of rows in a subarray	512, 1K, 2K, 4K, 8K, 16K
Number of columns in a subarray	256

Figure 2: Cache configuration parameters

We modified CACTI cache model version 2.0 [14] to estimate the timing and power consumption of different voltage scaled designs of two-stage pipelined caches. The cache parameters used in the analytical model are defined in Figure 2.

For the access method of the cache, a conventional parallel access read is used. By reading the N data ways in parallel with the tag array, the set-associative design allows the data and select signals for the data-selection multiplexor to arrive at nearly the same time. According to the timing of the circuit-level critical path of the cache, we split the circuit into two parts. The first pipeline component consists of the decoder, tag array and data array, while the second pipeline component contains the mux, sense-amplifier and comparator. A variety of timing models have been developed to estimate the delays of logic gates [15, 16]. These models are categorized as the simple RC model and its offspring, empirical delay models and fall/rise time based analytic models. We modified the RC delay model in [14] to a fall/rise time based analytic delay model in [15] so that we can evaluate the impact of the technology scaling factors such as supply voltage, threshold voltage, and transistor

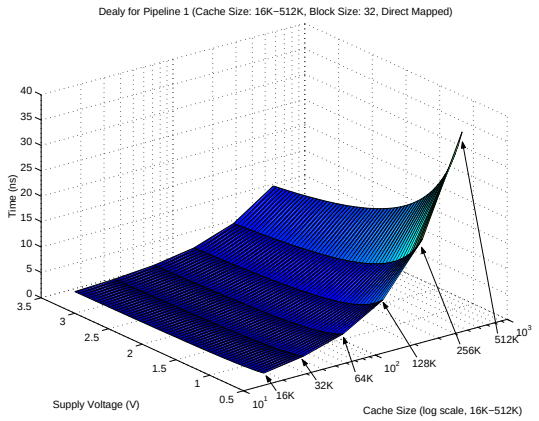

Figure 3: Delay for a cache pipeline stage 1

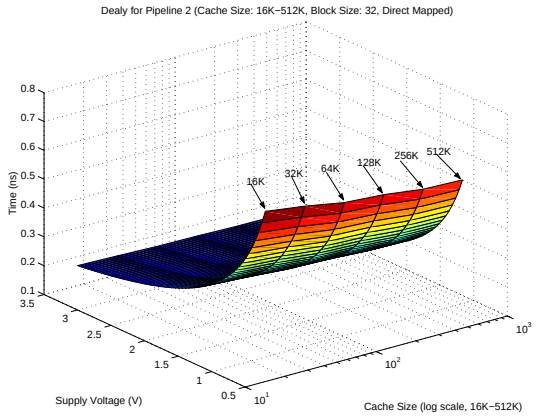

Figure 4: Delay for a cache pipeline stage 2

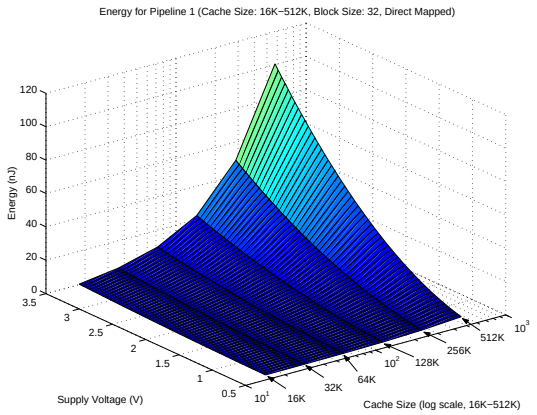

Figure 5: Energy for a pipeline stage 1

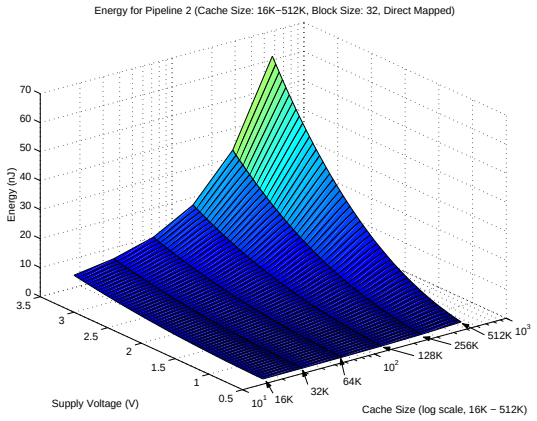

Figure 6: Energy for a cache pipeline stage 2

width. CACTI 2.0 has a detailed model of the wire and transistor structure of on-chip memories and provides very detailed capacitance values for each circuit component which is verified by Hspice. By using the capacitances, only the switching power component is estimated for each component at the circuit level. Throughout the simulations, the threshold voltage maintains a quarter of the supply voltage so that the delay due to the threshold voltage is not suffering excessively. Figure 3 and Figure 4 show the delay results of the two pipeline stages with different supply voltages and cache sizes, respectively. The delay of pipeline stage 1 increases as the cache size increases and the supply voltage decreases. However, the pipeline delay of stage 2 increases only if supply voltage decreases, because the critical path of pipeline stage 2 is not affected by the cache size. Figure 5 and Figure 6 show the energy dissipation results of the two pipeline stages with different supply voltages and cache sizes, respectively. The energy dissipation of pipeline stage 1 and pipeline stage 2 increases as the cache size increases and decreases as the supply voltage decreases.

4.3 Optimization of Energy and Delay

The optimized supply voltage for a cache is decided such that the worst-case pipelined cache stage delay is less than or equal to the cycle time (which is determined by the worst case processor pipeline stage delay). If a cache is evenly divided into two parts, each pipelined cache stage will potentially have an extra half cycle of time available versus when the entire cache execution had to occur in one clock cycle. This extra time available will be used for energy reduction.

Example 1. Consider a cache shown in Fig 7. In the base case, the stage cycle time is 10 ns and Vdd is 2.75 V. The pipelined cache for high performance has two pipeline stages, and each stage has a half of a base case cycle time. Therefore, new cycle time is 5 ns, but Vdd is still 2.75 V. It means the total energy consumed is same with the base case. In pipelined cache for low-power, it uses 10 ns cycle time instead of 5 ns in each pipeline stage. Therefore, there is 5 ns idling time. If the minimum supply voltage Vdd is 1.6 V (increasing the pipeline stage delay to 10 ns), then a 66% cache energy reduction can be achieved.

The optimal supply voltage can be chosen by the algo-

Table 1: Benchmarks (base case)

Benchmark	Description	Number of Inst.	I. Cache Access	D. Cache Access	Branch Misprediction	Load Use
sort_int	Sorting Integer	1721	1921	458	177	201
matmul	Matrix multiply	6268	7158	2254	604	512
arith	Arithmetic	2882	3755	509	105	151
factorial	Factorial	12033	16053	4006	4	1002
fi b	Fibonacci	2852	3664	1064	125	178

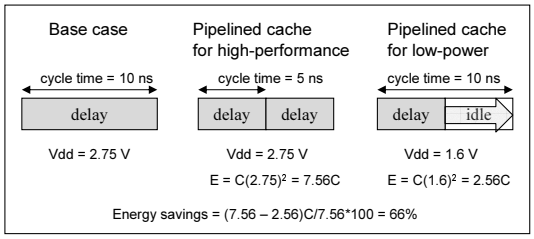

Figure 7: Illustration of energy of a pipelined cache

Voltage Optimization procedure for pipelined cache

```
Input: Vdd Range, delay_base
Output: Power optimal Vdd
Vdd Range ← [2.75v − 0.6v]
Vdd(0)=MAX(Vdd Range);
For i steps do
    Calculate delay_stage1(Vdd(i));
    Calculate delay_stage2(Vdd(i));
    If MAX [ delay_stage1{Vdd(i)}, delay_stage2{Vdd(i)} ] < delay_base
        Vdd_optimal = Vdd(i);
    endIf
    Dcrease Vdd(i);
endFor
```

Figure 8: Procedure for optimal supply voltage

rithm shown in Figure 8. The procedure chooses the supply voltage from the Vdd range in decreasing manner. The delay_base is the base case delay of a non-pipelined cache stage. The procedure calculates delay_stage1($Vdd(i)$) and delay_stage2($Vdd(i)$) corresponding to the chosen voltage $Vdd(i)$. If the larger delay value of the two stage delay values is smaller than delay_base, replace $Vdd\_optimal$ with $Vdd(i)$. Then the procedure decreases the $Vdd(i)$ and performs the steps again. Using this procedure, we find the optimal supply voltage that can minimize stage power within the stage delay criteria.

5 Results

The base case processor uses 2.75 V supply voltage for processor core and caches. We divided a cache into two parts. Table 2 shows Vdd, delay and energy of the base case and pipelined cache with different cache sizes. The Vdd of the pipelined cache is energy optimized supply voltage that can be achieved when the maximum of delay1 and delay2 is extended to delay of base case. The energy column is the energy consumed when the cache is accessed once. The delay1 of the pipelined cache increases when the cache size increases, but delay2 does not increase. The reason is that the cache stage 1 consists of a tag array and a data array, two main cache size dependent components. This heavy biased delay values are the reason why Vdd increases according to the cache size. A 16KB cache can save 69.60% of cache energy. But caches larger than 64KB can save smaller amount of energy than 16KB and 32KB caches, because the difference of Vdd of base case and Vdd of pipelined cache is not too much for a large size cache.

Table 1 is a list of benchmarks used and their characteristics. The number of instruction cache accesses and data cache accesses are used to estimate cache energy consumption using CACTI. The branch misprediction and load use count come from base case simulation. Those are important factors that affect the execution time of a pipelined cache processor.

The benchmarks are executed in the original and modified MARS models to obtain the processor core power values, benchmark execution times and cache access statistics. Table 3 shows execution time and processor core power used by the base case (original MARS) processor and the pipelined cache (modified MARS) processor. The misprediction and load use columns of Table 3 are the dynamic branch misprediction count and dynamic load use count respectively.

As mentioned above, these two cases carry extra execution time penalties when pipelined caches are used. In the sort_int and matmul benchmarks, execution times increase by 18.31% and 16.36%, respectively. Both benchmarks have a dynamic instruction count consisting of approximately 10% load use cases that cause extra stall cycles. To avoid these heavy penalties, dynamic or static scheduling schemes can be used. A dynamic scheme requires out-of-order instruction execution. Static scheduling is performed by a compiler that optimizes instructions to avoid load use cases. In our experiment, no special scheduling scheme was used to reduce the load use penalties for the seven-stage processor case (with pipelined caches). The misprediction of branches in sort_int also takes a huge portion of the execu-

Table 2: Energy delay for a pipelined cache

	Base case			Pipelined cache processor				
Cache(KB)	Vdd(V)	Delay(nS)	Energy(nJ)	Delay1(nS)	Delay2(nS)	Vdd(V)	Energy(nJ)	% Reduction
16	2.75	0.648	5.689	0.438	0.210	1.6	1.729	69.60
32	2.75	1.021	9.019	0.814	0.206	2.0	4.534	49.37
64	2.75	1.741	15.357	1.540	0.201	2.3	10.450	31.95
128	2.75	3.190	27.942	2.991	0.199	2.5	22.767	18.52
256	2.75	6.254	54.605	6.060	0.195	2.65	50.442	7.62
512	2.75	12.224	105.477	12.030	0.194	2.7	101.422	3.84

Table 3: Execution Time of base case and pipelined cache (Instruction Cache = 16KB, Data Cache = 16KB)

			Base case		Pipelined cache processor		
Benchmark	Misprediction	Load use	E.T(ns)	Core Power(mW)	E.T(ns)	Core Power(mW)	E.T.% Increment
sort_int	177	201	26595	1002	31465	1008	18.31
matmul	604	512	90485	1114	105293	1121	16.36
arith	105	151	43765	1079	47987	1086	9.65
factorial	4	1002	192345	981	221196	987	15.00
fib	125	178	40635	1057	47719	1063	17.43
average							15.35

tion time. There are at least two different approaches to hide branch penalty. We used a hardware based approach using a BTB. Another approach is a software based approach, such as backward taken and forward not taken scheme [8]. As shown in Table 3, the processor core power does not change much for the 5 stage versus the 7 stage processor. This means that the data path is nearly the same and is almost always busy. Therefore, the variation in total processor power between the two cases is heavily dependent on the cache power.

Both the 5-stage (original MARS) and 7-stage (modified MARS) processors each have a 16KB instruction cache and a 16KB data cache. The 5-stage base case uses a 2.75 V supply voltage (which is the default voltage assumed by the standard cell library we used) for the processor core and caches. The pipelined cache processor uses 2.75 V for the processor core and 1.6 V for the two pipelined caches. The Vdd chosen for the pipelined caches comes from Table 2. Table 4 and Table 5 show total power and energy distributions of each component with the various cache sizes. In the results, we do not consider the overhead for the dual supply voltage generator. This is because we assume that our target board already has both voltages available (for a practical example, the "Skiff" Personal Server Board already has multiple supply voltages[17]). The 16KB cache consumes 69.60% less energy when a reduced supply voltage of 1.6 V is used as compared to 2.75 V (note that this lower voltage is also applied to the I/O pads). The end result is that our pipelined cache processor consumes approximately 25% less processor power than the 5-stage base case. However, in the case of sort_int, only 9.70% less total processor (core + L1 caches) energy is used because processor core energy consumed increases for the 7-stage processor due to increased execution time.

From the experimental results, the branch misprediction penalty and load use penalty heavily affect both execution time and energy consumption. Therefore, to maximize the energy savings using a pipelined cache, a precise branch prediction scheme and an instruction scheduler that can reduce load use penalties must be introduced.

6 Conclusion and Future Work

With pipelined L1 caches supplied by a lower voltage, we showed three benchmarks where energy consumed is reduced by 10-15%. With additional micro-architectural and compiler support, we believe that we can improve the results to around 20-25% on each benchmark. Furthermore, we believe that the approach presented in this paper can be used in any case where cache-like memory is used and low power is an important issue.

To date we have only considered pipelined caches with two stages. In the future, we will investigate the effects of adding even more stages to the caches with various cache sizes. To more fully take advantage of the low power potential of pipelined caches, we need to apply advanced architectural techniques that have improved branch prediction and an instruction scheduler (compiler) that can better hide load use delay.

Acknowledgments

This research was funded by DARPA under contract number F30602-00-2-0564. We also acknowledge donations received from Cadence, Hewlett-Packard, Intel, LEDA Systems, Mentor Graphics, Sun and Synopsys.

Table 4: Power distribution of base case and pipelined cache (Instruction Cache = 16KB, Data Cache = 16KB)

	Base case (mW)				Pipelined cache processor (mW)				
Benchmark	Core	I. Cache	D. Cache	Total	Core	I. Cache	D. Cache	Total	% Reduction
sort_int	1002	411	98	1511	1008	120	25	1154	23.67
matmul	1114	450	142	1706	1121	134	37	1292	24.27
arith	1079	488	66	1634	1086	154	18	1258	22.96
factorial	981	475	118	1574	987	143	31	1161	26.24
fi b	1057	513	149	1719	1063	151	39	1253	27.09
average									24.85

Table 5: Energy distribution of base case and pipelined cache (Instruction Cache = 16KB, Data Cache = 16KB)

	Base case (nJ)				Pipelined cache processor (nJ)				
Benchmark	Core	I. Cache	D. Cache	Total	Core	I. Cache	D. Cache	Total	% Reduction
sort_int	26660	10929	2606	40195	31715	3789	794	36298	9.70
matmul	100830	40723	12823	154377	118034	14118	3898	136050	11.87
arith	47238	21363	2896	71496	52105	7406	880	60392	15.53
factorial	188628	91328	22791	302747	218220	31662	6928	256810	15.17
fi b	42953	20845	6053	69851	50743	7227	1840	59810	14.38
average									13.33

References

[1] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural level power analysis and optimizations," in *27th annual International synposium on Computer Architecture*, 2000.

[2] J. Montanaro and et. al., "A 160-mhz, 32-b, 0.5-w cmos risc microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1703–1714, 1996.

[3] L. T. Clarl and et. al., "An embedded 32-b microprocessor core for low-power and high-performance applications," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1599–1608, November 2001.

[4] V.G. Moshnyaga and hiroshi Tsuji, "Cache energy resuction by dual voltage supply," in *Proc. Int. Symp. Circuit and System*, 2001, pp. 922–925.

[5] M Igarashi and et al., "A low-power design methid using multiple suply voltages," in *International Symposium on Low Power Electronics and Design*, 1997.

[6] T.I Chappell, B.A. Chappell, S.E. Schuster, J.W. Allan, S.P. Klepner, R.V. Joshi, and R.L. Franch, "A 2-ns cycle, 3.8-ns access 512-kb cmos ecl sram with a fully pipelined architecture," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 11, pp. 1577–1585, 1991.

[7] J.L Hennessy and D.A. Patterson, "*Computer Architecture: A Quantitative Approach Second Edition"*, Morgan Kaufmann, San Francisco, California, 1996.

[8] K. Olukotun, T. N Mudge, and R. B. Brown, "Multilevel optimization of pipelined caches," *IEEE Transactions on computer*, vol. 46, no. 10, pp. 1093–1997, October 1997.

[9] The SimpleScalar-Arm power modeling project, http://www.eecs.umich.edu/~jringenb/power.

[10] Synopsys Inc., http://www.synopsys.com.

[11] J. E Smith, "A study of branch prediction strategies," in *Proc. 8th Annual International Symposium Computer Architecture*, 1981.

[12] LEDA Systems, Insc., http://www.ledasys.com.

[13] P. Korkmaz, K. Puttaswamy, and V. Mooney, "Energy modeling of a processor core using synopsys and of the memory hierarchy using the kamble and ghose model," in *Technical Report CREST-TR-02-002*. Georgia Institute of Technology, February 2002.

[14] G. Reinman and N. Jouppi, Cacti version 2.0, http://www.research.digital.com/wrl/people/jouppi/CACTI.html.

[15] N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, Santa Clara, California, 1992.

[16] J. Rubenstein, P. Penfi eld, and M.A. Horowitz, "Signal delay in rc networks," *IEEE Transactions on CAD*, vol. 2, no. 1, pp. 202–211, 1983.

[17] HP Cambridge Research Laboratory Personal Server Project, http://crl.research.compaq.com/projects/personalserver/personal-server- spec.html.