A DVS-based Pipelined Reconfigurable Instruction Memory

Zhiguo Ge, Tulika Mitra, and Weng-Fai Wong
School of Computing
National University of Singapore
{gezhiguo, tulika, wongwf}@comp.nus.edu.sg

ABSTRACT

Energy consumption is of significant concern in battery operated embedded systems. In the processors of such systems, the instruction cache consumes a significant fraction of the total energy. One of the most popular methods to reduce the energy consumption is to shut down idle cache banks. However, we observe that operating idle cache banks at a reduced voltage/frequency level along with the active banks in a pipelined manner can potentially achieve even better energy savings. In this paper, we propose a novel DVS-based pipelined reconfigurable instruction memory hierarchy called PRIM. A canonical example of our proposed PRIM consists of four cache banks. Two of these cache banks can be configured at runtime to operate at lower voltage and frequency levels than that of the normal cache. Instruction fetch throughput is maintained by pipelining the accesses to the low voltage banks. We developed a profile-driven compilation framework that analyzes applications and inserts the appropriate cache reconfiguration points. Our experimental results show that PRIM can significant reduce the energy consumption for popular embedded benchmarks with minimal performance overhead. We obtained 56.6% and 45.1% energy savings for aggressive and conservative V_{DD} settings, respectively, at the expense of a 1.66% performance overhead.

Categories and Subject Descriptors

B.3.2 [**Design Styles**]: Cache memories—*Memory structures*

General Terms

Algorithm, Design, Performance

Keywords

Instruction cache, low power, reconfigurable memory.

1. INTRODUCTION

Power consumption has become a major design concern with the proliferation of portable consumer electronics devices. As instructions are fetched in nearly every processor cycle, the instruction delivery system accounts for a significant fraction of the total energy consumption. Pose et al. [3] reported that the front-end instruction delivery path (consisting of the fetch, decode, rename, dispatch and issue stages) consumes about 20% of the overall system power and about 35% power of each processor core. Brooks et al. [4] found that instruction fetch consumes 22.2% of power in the Intel Pentium Pro processor. According to [15], the instruction cache of the StrongARM 110 processor accounts for 27% of total power and is higher than the energy consumption of the data cache.

Several approaches have been proposed to reduce the energy consumption of caches in embedded processors. For example, application specific customizations allow the cache parameters, such as associativity, line size, cache size, to be adapted to specific applications and/or their execution phases [1, 18, 7]. Dynamic voltage scaling (DVS) [6] is another effective technique for power reduction where the supply voltage and clock frequency are dynamically adjusted in accordance to the need of the application.

DVS-based techniques are especially useful in applications that have multiple phases of varying resource requirements. Resources that are under-utilized at a certain phases can be exploited to further reduce energy consumption. There are two different classes of techniques for doing this. The first class of approaches are *reactive* in nature in that they use DVS to slow down or completely shut down a resource when it becomes under-utilized. The second category of approaches *predict* the resource utilization and employ DVS to exploit the idle capacity of the under-utilized resources to save energy while maintaining performance. Proposals for each of these two methods have been made for computational units [13, 2, 9]. For example, an aggressive technique proposed in [6] replicates a logic block number of times with each instance operating at a lower supply voltage and frequency. In the context of memory hierarchies, most of the previously proposed techniques are reactive in nature. Examples include selective cache ways [1], drowsy cache [11] and cache decay [10] mechanism. They generally disable the under-utilized memory resources to save energy.

This paper presents a novel predictive DVS-based energy savings technique for the memory hierarchy that uses idle banks rather than shutting them down. We propose the *pipelined reconfigurable instruction memory hierarchy* (PRIM) for power constrained embedded processors. Our canonical example of PRIM consists of an instruction cache with four banks for data storage, where two banks can be dynamically reconfigured to DVS mode. In DVS mode, the supply voltage and frequency of the two banks are decreased. Moreover, the low-voltage banks are operated as scratch-pad memory that are explicitly controlled through software. In particular, frequently executed instructions are loaded and locked into the two DVS banks. Finally, instruction throughput is maintained by pipelining and alternating the instruction fetches between these two banks. To the best of our knowledge, this is the first work

DAC 2009, July 26 - 31, 2009, San Francisco, California, USA.

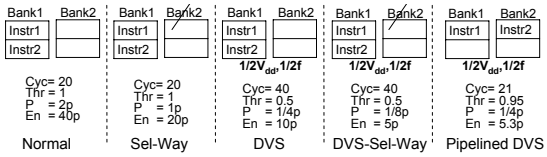

Figure 1: Comparison of cache energy savings techniques.

that proposes taking advantage of under-utilized storage resources to obtain more energy savings instead of disabling them.

Motivating Example. Let us illustrate the intuition behind PRIM and its advantage over existing cache energy savings techniques with a motivating example. Consider the execution of a small loop containing only two instructions from a 2-way set associative cache for 10 iterations. For simplicity of exposition, let us assume that each cache block contains only one instruction. Finally, both the instructions are assumed to be present in the same cache bank. Figure 1 shows a comparison among the different schemes based on (1) total cycles for instruction fetch (*Cyc*), (2) instruction fetch throughput (*Thr*), (3) power (*P*) and (4) total energy consumption (*En*) for one complete execution of the loop. We assume 1-cycle cache latency in normal mode and 2-cycle cache latency in DVS mode (as DVS mode runs at half the frequency). The dynamic power of each normal cache bank is $p = CV^2f$.

In normal mode, it will take 20 cycles to fetch 20 instructions with a throughput of 1 instruction/cycle. As the normal cache has two banks, $P = 2p$ and $En = 2p \times 20$. In the *selective-way cache*, Bank2 is switched off as all the accesses go to Bank1. So the selective-way scheme can achieve 50% power and energy reduction compared to normal cache without sacrificing performance.

A more aggressive technique applies *DVS* to both the cache banks. The power requirement of DVS scheme is $P = 2 \times C \times (\frac{V}{2})^2 \times \frac{f}{2} = \frac{p}{4}$. However, as frequency is halved, the instruction fetch latency increases to 2 cycles and throughput drops to 50% of the normal cache. The overall energy consumption of DVS scheme is $40 \times (\frac{p}{4})$, which is 50% savings compared to selective-way cache. To save additional power, the unused bank can be switched off in DVS mode as well leading to *DVS selective way* cache. Clearly, DVS selective way has the same performance as DVS scheme but reduces energy consumption further by 50%. Our key observation is that in DVS mode, one cache bank is not utilized and it either sits idle or gets switched off. Instead of powering down one cache bank for this loop, PRIM (a) distributes the instructions to the two cache banks and (b) powers up both the cache banks in DVS mode. We then propose the use of pipelined instruction fetches to hide the latency introduced by DVS thereby achieving the one instruction fetch per cycle throughput of the normal cache. An example of this pipelined access is shown next.

Cycle :	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	...
Bank1 :	$i1$		$i1$		$i1$	...	
Bank2 :		$i2$		$i2$		$i2$	...

Clearly, the throughput of 1 instruction per cycle is sustained. However, an accurate next instruction address prediction scheme needs to be in place in order for this to succeed.

In summary, PRIM has the same power consumption as the DVS scheme. However, the careful overlapping of accesses to the two banks reduces the total cycles requirement to 21 (1 additional cycle is required for pipeline initialization). Thus the total energy consumption for this scheme is $En = \frac{p}{4} \times 21 = 5.3p$, which is similar to DVS selective way scheme (the best scheme in terms of energy) while the performance is similar to that of the normal cache. Moreover, with larger loop sizes and greater number of iterations, the pipeline initialization overhead will become negligible.

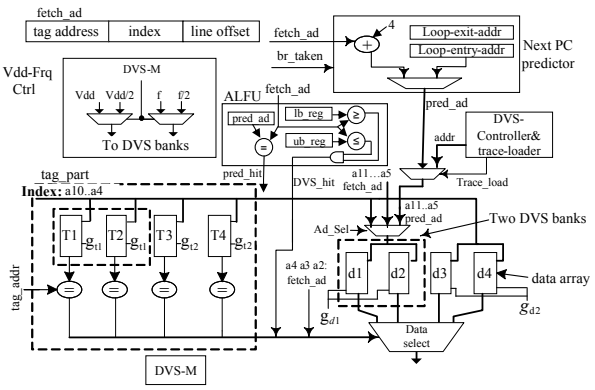

Figure 2: PRIM architecture.

The motivating example illustrates that PRIM needs co-operation from the compiler in statically identifying program phases with under-utilized cache banks. Once these program fragments have been identified, the compiler inserts appropriate instructions to load and lock the content to the cache banks and switch the banks to DVS mode. At runtime, the execution follows these cache configuration hints to switch the cache between normal and DVS modes.

2. THE PRIM ARCHITECTURE

Architectural considerations: We present the PRIM architecture based on a 4-way set associative cache as shown in Figure 2. PRIM architecture can achieve the following functionalities: (1) run in DVS mode and normal cache mode according to execution needs, (2) dynamically switch running modes, (3) reload instruction blocks to DVS banks at runtime, and (4) fetch instructions from DVS banks in pipelined fashioned to maintain throughput. Here we only consider uninterrupted execution of a single application that controls PRIM throughout its execution.

Control of DVS mode and normal cache mode: The cache consists of four data banks, the DVS control logic, the voltage-frequency controller (Vdd-Frq Ctrl), the address lookup function unit (ALFU), and the next-PC predictor. Unlike a conventional cache, the supply voltage of banks d1 and d2 can be dynamically changed by the DVS controller. When a DVS reconfiguration instruction is encountered, the DVS controller loads the appropriate instructions to the DVS banks. DVS controller is essentially a component of Direct Memory Access (DMA) which is controlled by processor. It has two inputs: initial address of an instruction block and number of instructions to be loaded. After loading the instructions, DVS controller sets the DVS-M register, which in turn changes the supply voltage and frequency of d1 and d2 to $\frac{V_{dd}}{2}$ and $\frac{f}{2}$, effectively reconfiguring the banks to DVS mode.

Instruction searching: The ALFU (*address lookup functional unit*) determines if an instruction is residing in the DVS banks or not. When two banks are configured to DVS mode, the tags corresponding to these two banks are completely disabled. In other words, the DVS banks act as software controlled memory. The address matching is accomplished inside the ALFU instead of tag matching in conventional caches. The ALFU contains two address registers and two parallel comparators. The two registers, `ub_reg` and `lb_reg`, hold respectively the upper and lower bound addresses for the instruction block that currently resides in the DVS banks. The values in `ub_reg` and `lb_reg` are updated when the DVS controller loads a block of instructions into the DVS banks. In the current PRIM design, there is only one pair of `ub_reg` and `lb_reg`. As a result, only one block of instructions may reside in the DVS banks at any point of time. If the address of the instruction

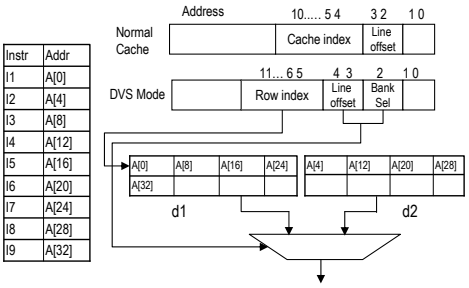

Figure 3: The layout of instructions

to be fetched falls within the range of these two registers, then it resides in the DVS banks and the ALFU generates `DVS_hit` signal. This signal controls the selection and gating of the tag and the data banks. To save dynamic energy, the two normal cache banks and their tags are not searched if the `DVS_hit` signal is asserted. On the other hand, the DVS banks are not searched if `DVS_hit` signal is not asserted. This is accomplished by clock gating of the tag and data banks. The gating signals g_{t1}, g_{d1} and g_{t2}, g_{d2} are used to gate the clock of DVS banks and non-DVS banks, respectively:

$$g_{t1} = \text{DVS\_mode} \qquad g_{t2} = \text{DVS\_mode} \wedge \text{DVS\_hit}$$
$$g_{d1} = \text{DVS\_mode} \wedge \sim \text{DVS\_hit} \qquad g_{d2} = \text{DVS\_mode} \wedge \text{DVS\_hit}$$

As a result, in DVS mode, only two banks – either DVS banks or normal cache banks – are accessed at any point of time.

Pipelined parallel instruction fetches: In order to maintain the throughput of instruction fetches in DVS mode, when an instruction is fetched from one DVS bank, a speculative fetch is also started in the other DVS bank. The ALFU will later check to see if the speculation was successful. If the speculated PC equals that requested by the processor, the address prediction hit (pred_hit) is asserted to indicate a successful speculation. Otherwise, it is necessary to redo the instruction fetch, thereby incurring an overhead of twice the normal cache's latency.

>From the above description, one can easily see why it is crucial to carefully layout the instructions in DVS banks so that sequentially executed instructions can be fetched in parallel from the two DVS banks. Figure 3 shows an example of the instruction layout. The left side of the figure is the instruction layout of a loop containing six instructions in main memory. Suppose the sequence of instruction executed is $1-2-3-4-5-6-1\cdots$. To be able to fetch consecutive instruction in parallel, the odd and even numbered instructions need to be in different banks. We propose to layout the instructions in the way shown in right hand side of Figure 3. Bits 11...5 in Figure 3 are used to address the row of the DVS banks, while the column selection is determined by the least significant bit (bit 2 in Figure 3) of cache block index. The instruction selection from a cache line is decided by two bits 3 and 4.

As the two DVS banks can be configured as two different types of instruction buffer, namely conventional cache and software controlled memory, the address of two DVS banks need to be selectable from different address sources according to their mode. This is achieved by selection signal Ad_Sel. When applications try to load instruction block into DVS mode, Ad_Sel will select the address generated by 'DVS-Controller&trace-loader' component. If they are configured as DVS mode and operate on instruction fetching state, address a11...a5 will be chosen. Otherwise, the address (i.e cache index) will be selected for conventional cache mode.

The 'Next PC predictor' component of PRIM predicts the address of the next instruction to be fetched. If a branch is taken, we assume that we have encountered a loop and the address boundary will be dynamically written to registers 'Loop-entry-addr' and 'Loop-exit-addr'. If the current fetch address is not equal to the value in 'Loop-exit-addr', the next fetch address will be speculated to be the increment of the current fetch address. Otherwise, the next fetch address will be the value in 'Loop-entry-addr'.

3. COMPILER SUPPORT

PRIM requires compiler support to achieve dynamic reconfiguration. The compiler decides which program regions are suitable for running in DVS mode, inserts appropriate instructions into the application to switch the two designated banks to DVS mode, and dynamically loads the selected instructions into the DVS banks.

Our algorithm works on the Loop-Procedure Hierarchy Graph (LPHG) [14] of the program. The nodes of this graph represent procedures and loops while the edges between the nodes denote the call relationship between them.

Algorithm 1: Algorithm for determining dynamic reconfiguration and DVS instruction load points

Input: $Proc\_list$: Procedure list whose procedures have been intra-procedural optimized
Output: $Basic\_block\_list$: list of instruction blocks of basic blocks assigned to DVS banks

```
   BOOL conflict : if severe cache conflict incurred;
1  Build Loop-Procedure Hierarchy Graph(LPHG)
2  Get_sizes_of_all_loops();
3  list_loops ⟵ all leaf loops;
4  foreach loop l in list_loops do
       if l is leaf loop ⋀ #iterations of l ≥ Thresh_hold then
           Annotate_reconfig_point_and_instrs_partitioned(l);
5      else if l is non-leaf loop then
6          list_child_loops ⟵ all child loops of l;
7          Update_iteration_num(list_child_loops);
8          conflict=evalConflict(DVSmode, child_loops_of_l ∪ l);
9          if !conflict then
10             Instr_alloc(list_child_loops ∪ l, DVS-banks);
           end
11     if !list_loops.contain(l) then
12         list_loops.push_back(l);
       end
   end
13 Hoist_reconfig_position();
14 Insert_reconfig_and_code_loading_instructions();
15 return Proc_list;
```

Our proposed algorithm is shown in Algorithm 1. We assume that most of the energy consumption due to instruction fetch occur inside the loops. The intuition is that if the time spent in a loop is long enough to hide the overhead of reconfiguration and instructions loading, then that loop is a candidate for allocation into the DVS banks. Thus we look for loops where the number of iterations exceeds a threshold. In this paper, we empirically set the threshold value to be 20 iterations. If the loop size is too big to fit into the DVS banks, then the cache is used to buffer the rest of for the loop.

In a LPHG, the deeper a loop is, the higher is its execution frequency. The algorithm therefore starts from the leaf loops and works toward the root node. After a loop has been examined, its parent loop is added to $list\_loops$ (line 12). We may at some later point examine the parent node for further opportunities. If a loop is an internal node (line 5), then the algorithm evaluates whether it is beneficial to configure the cache to DVS mode for its child loops (line 8). The evaluation function we use is conservative and simple. If the allocation of a child loop L to DVS banks does not adversely affect the miss rate of the other children, then the allocation is considered beneficial. In that case, the cache is configured to DVS mode at the entry point of the parent loop.

Figure 4 is an example of how the algorithm resolves conflicts, determines reconfiguration points, and selects the instructions for

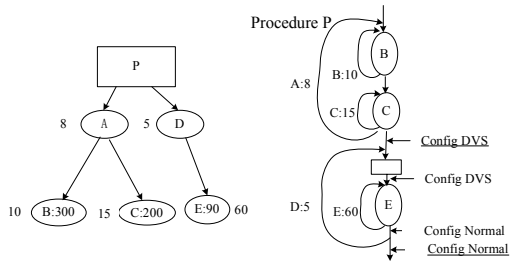

Figure 4: Code transformation for reconfiguration.

DVS banks. The left part of Figure 4 is a sample program represented in LPHG, while the right part is its corresponding CFG. The LPHG consists of the procedure P and five loops A, B, C, D, E. The numbers beside the loops are the number of loop iterations while the numbers inside the loops are their sizes in terms of the number of instructions. Each of the four cache banks can hold up to 64 instructions. The algorithm first selects the three leaf loops. Only the number of iteration of E is larger than the threshold value (20). So the cache is configured to DVS mode at the entry point of E. We then traverse upward to the parent loops, A and D. As the number of iterations of D is smaller than the threshold, we will not use DVS mode for it. Loop A has two children, B and C, and the total number of iterations of B and C from the point of the entry of A is now larger than 20. The algorithm then checks whether there will be severe cache conflicts if B or C is placed in the DVS banks. As the sizes of B and C are both larger than the remaining two normal cache banks, allocating either loop to the DVS banks will cause severe cache conflict to the other loop. It is therefore prudent to operate A in DVS mode.

The instruction allocation function assigns the frequently executed instructions inside the selected loop to the DVS banks (line 10). If the loop size exceeds the capacity of the DVS banks, then only the most frequently executed instructions are allocated to DVS banks.

After determining the reconfiguration points and the instructions selection for the DVS banks, the number of reconfigurations can be reduced by hoisting the reconfiguration point from the inner loop to the outer loop (line 13) if L is the only loop selected among its siblings. Figure 4 shows an example where the DVS configuration instructions are hoisted to where they are underlined.

4. EXPERIMENTAL EVALUATION

4.1 Experimental Methodology

We used the Simplescalar-PISA 3.0d simulator [5] for our experiments. The full-featured simulator `sim-outorder` was modified to support PRIM. The cache line used in the simulation corresponds to 32 bytes. The instruction memory hierarchy consists of PRIM and the main memory. Our PRIM implementation has four banks of data storage, each capable of holding 64 instructions. The latency of accessing L1 cache is 1 cycle. The main memory is assumed to be pipelined. The latency of the first access to the main memory is 10 cycles, while that of the subsequent accesses is 2 cycles. In our simulation, the Trace_load instructions used to load instruction blocks to DVS banks are compiled to binary code. We include the overhead of execution time and cache miss rate increase caused by these instructions.

We use six benchmarks from the MediaBench and MiBench suites. We compared the energy consumption and performance across three different configurations: (1) a baseline system comprising of a traditional 4-way associative instruction cache, (2) a simple low V_{DD} scheme (Low-V_{DD}) (3) a selective way cache [1] (Sel-Way), and (4) PRIM. We use two voltage settings, $0.5\times V_{DD}$ and $0.8\times V_{DD}$, for different process technologies. We chose selective way cache for comparison because it disables and puts under-utilized memory banks into low power mode to save energy.

In our experiments, Sel-Way gates the clock of a certain number of cache banks if it is under-utilized. To do this, we analyze the LPHG and put frequently executed loops into certain cache banks and gate the rest of the cache banks. The instructions of the frequently executed loops are locked for a certain time interval to avoid unintentional replacement. Low-V_{DD} scheme simply halves supply voltage and clock frequency of L1 instruction cache to save energy while the access latency to the instruction cache is doubled.

4.2 Energy Calculations

We model the energy consumption of the memory hierarchy using the CACTI [17] model for $0.13\mu m$ technology. In this work, our focus is on reducing dynamic energy consumption. To calculate the energy consumption of PRIM, we include the extra logic elements not found in a normal cache, namely the ALFU, the Next PC predictor and the two extra multiplexers. We assume that the energy consumed by the PRIM's comparators and multiplexers are the same as the multiplexers and comparators components of the normal cache assumed by CACTI. We approximate the energy consumption of an adder or subtractor in ALFU as that of a multiplexer. These hardware components are powered at regular voltage.

According to CACTI, the total per-access energy of a n way cache is $e_C = e_t + e_d + e_{com} + e_{mux} + e_{out}$ where e_t, e_d, and e_{com} stand for the energy per access to all tags, data storages, and comparators, respectively. e_{mux} is the energy consumption of the multiplexer, while e_{out} is the energy of the output driver. We rewrite this formula as $e_C = n \times e_b + e_{mux} + e_{out}$ where e_b is the energy per access for one cache bank. This includes the per-access energy for the data and tag of one bank, and a comparator.

According to general energy equation, $E = C\times V^2 \times f \times t$, when supply voltage and frequency is decreased to V', the energy consumption becomes $(\frac{V'}{V})^2 \times 2f$ (we define this factor as C_f) of the original value. We assume that the energy of the data output drivers of PRIM will be the same as that of a normal cache since this component has to be powered at the regular voltage while that for the cache banks, comparators and multiplexer is decreased. There are three ways in which PRIM is accessed. When PRIM is in DVS mode and the instruction is in one of the DVS banks, PRIM can deliver it directly from the DVS bank without searching the other DVS bank, the energy of an instruction fetch is: $e_D = \frac{e_b + e_{mux}}{C_f} + e_{out} + e_O$, where e_O is the energy overhead per access due to the extra hardware components described above. It is about 3.5% of the total per-access energy of the baseline cache. When PRIM is in DVS mode and the instruction is not in a DVS bank, the other two cache banks powered at the regular V_{DD}, will be searched. The per-access energy for this is: $e_{\neg D} = 2 \times e_b + e_{mux} + e_{out} + e_O$. If PRIM is in normal cache mode, energy consumption is simply the energy of the baseline cache plus the overhead: $e_N = e_C + e_O$.

The last two energy components of PRIM are the energy consumed during the loading of traces, E_L, into the DVS banks, and the energy caused by cache misses. E_L is approximated by:

$$E_L = \sum_{l=0}^{L} \frac{s_l}{B} \times e_M$$

where L is the total number of traces loaded, s_l is the size of trace l and B is the size of a cache block. e_M is the energy penalty of a

base cache(e_C)	Low-V_{DD}	PRIM				Sel-Way			
		$e_{\neg D}$	e_N	e_D (0.5×V_{DD})	e_D (0.8×V_{DD})	1way	2way	3way	4way
0.538	0.144	0.296	0.557	0.065	0.165	0.146	0.277	0.407	0.538

Table 1: Per-access energy consumption (in nJ).

cache miss. Its value is assumed 50 times of energy consumption of one access to four way baseline cache [18]. Since the DVS controller is only active during instruction trace loading to PRIM, and because the energy of a cache miss is orders of magnitude greater, we did not include it in the energy calculations. The total overall energy consumption due to memory accesses to PRIM is given by:

$$E_{\text{PRIM}} = A_D \times e_D + A_{\neg D} \times e_{\neg D} + A_N \times e_N + A_M \times e_m + E_L$$

where A_D and $A_{\neg D}$ are the numbers of accesses to the DVS banks and two cache banks respectively when PRIM is in DVS mode, while A_N is the number of accesses to PRIM when it is in normal cache mode. A_M is the number of cache misses.

As with PRIM, for Low-V_{DD} cache, the voltage of data output driver is assumed normal and the energy consumption of this component remains same as that of baseline cache. For Sel-Way cache, the energy is depend on the number of active cache banks, n, which is shown as $e_S = n \times e_b + e_{mux} + e_{out}$

Table 1 shows the energy consumption per access to different architectures. The result of our energy calculation is consistent with the energy consumption of 32K L1 cache in [12] in which SPICE was used to simulate the energy consumption. According to [12], the total energy consumption of 32K L1 cache powere at 1V is 1.055nJ while the energy at 0.5V is 0.1250nJ.

4.3 Experiment results

Performance: The performance results are shown in Table 2. 'fetch miss' represents the miss rate of instruction fetch while 'perform overhd' stands for the performance overhead compared to the baseline. 'Pred-miss' is the miss rate of address prediction of the next speculatively fetched instruction in DVS mode. Compared to the baseline cache configuration, we can see some miss rate reduction achieved by PRIM and Sel-Way schemes for certain benchmarks. We attribute this to the locking of frequently executed instructions, thereby reducing conflict misses. The miss rate of address prediction ranges from 0.63% to 8.45% with an average value of 2.85%. The 'pred-miss' rate for mpeg2-dec and pegwit are quite high. The loops in mpeg2-dec have many procedure calls resulting in poor locality. For pegwit, there are many control flow transfers inside the loops which is detrimental to the prediction process.

The performance overhead varies across different applications. There are mainly four factors affecting the overhead: the latency of cache accesses, the cache miss rate, the next address prediction miss rate, and the overhead of cache reconfigurations. Low-V_{DD} suffers from a high performance overhead that averages 125.6% because of the doubling of the latency in the instruction cache. PRIM, on the other hand, has a low performance overhead ranging from -6.65% to 2.82%. For the benchmark susan-edge, performance is in fact improved because of the reduction in the cache miss rate. On the other hand, for the benchmark pegwit, the poorer performance is due to the penalty of pred-miss and reconfiguration exceeding the gains from cache miss rate reduction.

Energy consumption: The total dynamic energy consumption of the four instruction memory hierarchies are shown in Table 3. We evaluated two PRIM configurations – one that halves V_{DD} and one that lowers V_{DD} by 20%. In both these configurations, the operating frequency is halved, doubling access times to these banks. All the schemes achieve significant energy savings compared to the baseline. The reduction in energy consumption achieved by the Sel-Way cache is 39.0% for the benchmarks, while that for PRIM is 56.6% at 0.5×V_{DD} and 45.1% at 0.8×V_{DD} . In other words, PRIM at 0.5×V_{DD} and 0.8×V_{DD} achieved 17.6% and 6.1% more energy saving than Sel-Way respectively. Energy savings of PRIM at 0.8×V_{DD} is smaller than PRIM at 0.5×V_{DD} because the supply voltage scaling down is more conservative.

Low-V_{DD} achieved an average 56.5% energy savings which is almost the same as PRIM at 0.5×V_{DD} . However, Low-V_{DD} comes with significant performance loss as shown in Table 2 because of the doubling of the cache hit latency. As the result, the additional energy consumed by other parts of a processor can surpass the energy savings obtained by instruction memory hierarchy.

Energy savings come mainly from DVS mode access, clock gating of under-utilized ways, and cache miss rate reduction. Operating in the DVS mode, PRIM can save more energy than the clock gating used in Sel-Way. This is evident in the benchmark FFT where the improvement of PRIM over Sel-Way scheme is very significant. In the case of FFT, the cache miss rate is very low and so cache misses contribute only a small amount of total energy consumption. The main contributor to the total energy is the on-chip cache access energy which is significantly reduced in PRIM. For the benchmark susan-edge, the energy improvement of PRIM over Sel-Way is small. This benchmark has a big loop with a large number of iterations that is larger than the cache. As a result, it is beneficial to allocate as many cache entries as possible and lock the instructions of this loop in these cache entries to avoid cache conflict. Sel-Way allocates and locks three cache banks for the instructions in the loop, while our PRIM configuration can only allocate two cache banks in DVS mode. Sel-Way is therefore able to avoid more cache conflicts in this benchmark.

Energy delay product: 'En*delay' in Table 3 represents normalized energy delay product relative to the baseline cache. PRIM at 0.5×V_{DD} , PRIM at 0.8×V_{DD} , Low-V_{DD} and Sel-Way achieved average normalized energy delay product value 0.43, 0.54, 0.93, and 0.61 respectively. Although Low-V_{DD} achieves significant energy savings, the energy delay product is about the same as the baseline cache due to the large performance overhead. PRIM at 0.5×V_{DD} and 0.8×V_{DD} achieved the best energy-delay product since it can effectively decreased energy consumption with only a slight performance loss.

5. RELATED WORK

A lot of research have been done on the memory hierarchy with the aim of saving energy. Schemes were proposed to shut down certain number of cache ways when the cache is under-utilized [1, 16]. Zhang [18] proposed a highly reconfigurable cache whose associativity can be dynamically changed.

Many schemes and algorithms utilizing dynamic voltage and frequency scaling have also been developed for low energy processors. In [13, 2], the worst-case slack time and workload-variation slack time were exploited using DVS. When the utilization computed based on the WCETs of tasks is lower than 1, it may be possible to run some of tasks at a slower clock and a lower supply voltage without delaying their deadlines. Some researchers [2, 8] developed compilation flows that detect code regions for DVS

Benchmark	Baseline	Low-V_{DD}	PRIM			Sel-Way	
	fetch miss(%)	perform overhd(%)	fetch miss(%)	pred-miss (%)	perform overhd(%)	fetch miss(%)	perform overhd(%)
mpeg2-dec	1.35	107.0	1.30	4.8	-0.03	1.49	3.07
gsm-dec	0.42	121.8	0.40	0.71	0.20	0.41	0.07
susan-edge	2.76	80.9	2.36	1.11	-6.65	2.15	-13.5
FFT	0.06	146.5	0.09	1.42	2.82	0.09	0.85
pegwit	0.38	45.8	0.35	8.45	0.78	0.38	3.54
sha	0.054	252.1	0.35	0.63	1.66	0.046	3.85
Average	-	125.6	-	2.85	-	-	-

Table 2: Miss rate and performance overhead

	Baseline	Low-V_{DD} ($0.5\times V_{DD}$)			PRIM($0.5\times V_{DD}$)			PRIM($0.8\times V_{DD}$)			Sel-Way		
Benchmark	En(mJ)	En(mJ)	impr(%)	En*delay	En(mJ)	impr(%)	En*delay	En(mJ)	impr(%)	En*delay	En(mJ)	impr(%)	En*delay
mpeg2-dec	34.5	19.43	43.7	1.17	25.6	25.8	0.74	27.55	20.1	0.80	30.26	12.3	0.90
gsm-dec	8.10	3.20	60.5	0.88	3.66	54.7	0.45	4.61	43.0	0.57	4.35	46.3	0.54
susan-edge	2.92	2.02	30.8	1.25	1.89	35.2	0.60	2.00	31.4	0.64	1.98	32.3	0.59
FFT	1.06	0.31	71.0	0.71	0.21	80.0	0.21	0.40	62.0	0.39	0.51	51.5	0.49
pegwit	19.78	7.63	61.5	0.56	7.78	60.7	0.40	10.19	48.5	0.52	11.21	43.4	0.59
sha	7.53	2.16	71.3	1.01	1.25	83.4	0.17	2.58	65.8	0.35	3.89	48.4	0.54
Average	-	-	56.5	0.93	-	56.6	0.43	-	45.1	0.54	-	39.0	0.61

Table 3: Energy consumption.

and determine the optimal frequency scaling. There are also many proposals to apply DVS to the memory hierarchy. Kim et. al. [11] proposed the drowsy cache architecture where cache lines will be put into drowsy mode if they are not accessed for a certain time interval. This technique is quite similar to the selective way cache [1] in which under-utilized cache lines are set to a low power mode. The main difference between them is that the drowsy cache can hold data when switching between low and normal power modes. All these previous works tried to disable under-utilized memory resources to prevent these parts from consuming energy. Instead PRIM tries to make use of these resource in DVS mode so as to retain the energy savings while recovering the performance loss.

6. CONCLUSION

In this paper, we propose PRIM, a pipelined, DVS-based dynamically reconfigurable instruction memory hierarchy. A number of ways in a PRIM cache can be reconfigured as low-V_{DD} , low frequency loop buffer. This achieves better energy savings over methods that disable unused resources for energy reduction. Pipelined speculative reads are used to maintain the throughput of instruction fetch in the low-V_{DD} buffer. To support PRIM, we also developed an algorithm to partition instructions and determine reconfiguration points. Our experimental results showed significant energy savings while performance is affected only marginally.

7. REFERENCES

[1] David H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proceedings of MICRO*, 1999.

[2] Ana Azevedo et al. Profile-based dynamic voltage scheduling using program checkpoints. In *Proceedings of DATE*, 2002.

[3] P. Bose et al. Early-stage definition of LPX: A low power issue-execute processor prototype. In *Proceedings of HPCA Workshop on Power-Aware Computer Systems*, 2002.

[4] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of ISCA*, 2000.

[5] Doug Burger and Todd M. Austin. The Simplescalar tool set, version 2.0. *Technical Report #1342, University of Wisconsin-Madison Computer Sciences Department*, May 1997.

[6] A. Chandrakasan and R. W. Brodersen. Lower power digital CMOS design. In *Kluwer Academic Publishers*, 1995.

[7] Zhiguo Ge, Weng-Fai Wong, and Hock Beng Lim. DRIM: A low power dymaically reconfigurable instruction memory hierarchy for embedded systems. In *Proceedings of DATE*, 2007.

[8] Chung-Hsing Hsu, Ulrich Kremer, and Michael S. Hsiao. Compiler-directed dynamic voltage/frequency scheduling for energyreduction in microprocessors. In *Proceedings of ISLPED*, 2001.

[9] Zhigang Hu et al. Microarchitectural techniques for power gating of execution units. In *Proceedings of ISLPED*, 2004.

[10] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of ISCA*, 2001.

[11] Nam Sung Kim et al. Drowsy instruction caches: Leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proceedings of MICRO*, 2002.

[12] Cheng-kok Koh, Weng-Fai Wong, Yiran Chen, and Hai Li. VOSCH: Voltage Scaled Cache Hierarchies. In *Proceedings of ICCD*, 2007.

[13] Soongsoo Lee and Takayasu Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proceedings of DAC*, 2000.

[14] Yanbing Li et al. Hardware-software co-design of embedded reconfigurable architectures. In *Proceedings of DAC*, 2000.

[15] James Montanaro et al. A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor. *Digital Technical Journal*, 9(1), 1997.

[16] Michael Powell et al. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of ISLPED*, 2000.

[17] Steven J. E. Wilton and Norman P. Jouppi. CACTI: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, 1996.

[18] Chuanjun Zhang, Frank Vahid, and Walid Najjar. A highly configurable cache architecture for embedded systems. In *Proceedings of ISCA*, 2003.