Value Prediction Design for High-Frequency Microprocessors

Ravi Bhargava and Lizy K. John

Technical Report TR-020508-01
Laboratory for Computer Architecture
Department of Electrical and Computer Engineering
The University of Texas at Austin
{ravib,ljohn}@ece.utexas.edu

May 2002

Abstract

This paper studies value predictor design under table access latency and energy constraints for high-frequency, wide-issue microprocessors. Previous value prediction efforts make generous assumptions regarding table sizes and access conditions, while ignoring prediction latencies and energy issues. Our work shows that the latency of a high-performance value predictor cannot be completely hidden by the early stages of the instruction pipeline as previously assumed, and causes noticeable performance degradation versus unconstrained value prediction. This work describes and compares three varieties of value prediction strategies: At-Fetch, Post-Decode, and Decoupled.

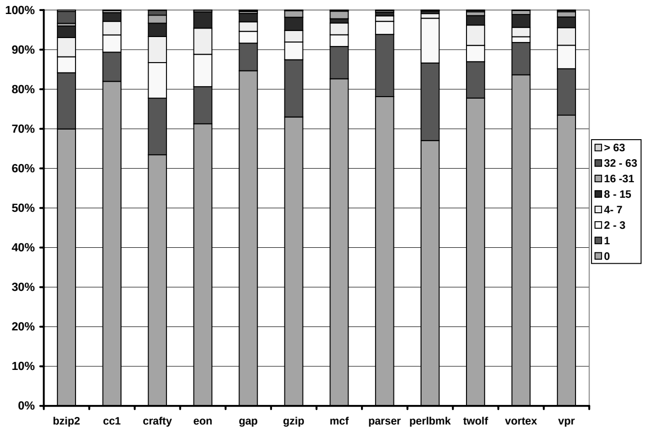
Decoupled value prediction excels at overcoming the high-frequency table access constraints but with high energy consumption requirements. A load-only value predictor can greatly reduce energy consumption but at the cost of performance. We propose a value prediction approach that combines the latency-friendly approach of Decoupled value prediction with a more energy-efficient implementation. The traditional PC-indexed prediction tables are removed and replaced by updates of buffered prediction traces. This latency and energy aware form of accounting for and distributing the value predictions leads to a 58%-95% reduction in dynamic energy versus common high-performance value prediction hardware while still maintaining high performance.

1 Introduction

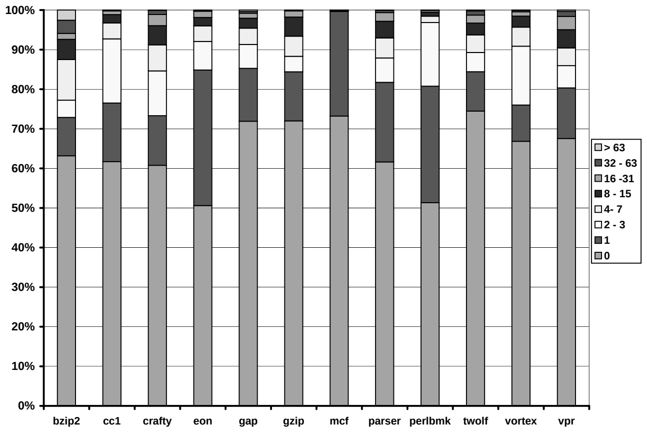
A trend in recent microprocessors design is the increasing density of on-die transistors. This trend has allowed designers and researchers to explore creative new techniques for improving instruction throughput. One strategy for utilizing this wealth of transistors is increasing execution resources and speculation hardware. Value prediction is one technique that has shown great potential for improving instruction throughput. True data dependency is a fundamental obstacle to higher instruction level parallelism (ILP), and value prediction speculatively satisfies true data dependencies early by predicting instruction results. This leads to higher utilization of the processor resources as well as faster resolution of slowly completing instructions.

Urgency To Predict Values: A successful value prediction is only beneficial when it successfully breaks true data dependencies, allowing instructions that are typically serialized to execute in parallel. This is especially useful in wide-issue microprocessors with plentiful execution resources, which are often underutilized.

One way to quantify the potential usefulness of successful value prediction is to study the distance between result-producing instructions and their consumers. In Figure 1, we measure the producer-consumer distance in clock cycles for a specific implementation of a wide-issue microarchitecture. The cycles are measured from instruction issue (placement in instruction window) to the first *request* from a consumer instruction (when placed in instruction window). These graphs measure the urgency to break data dependencies. Notice that 78-94% of load instructions have a consumer within one cycle, and 73-99% of integer instruction results are in demand within one cycle. Consumers appear quickly in a wide-issue microarchitecture capable of fetching past multiple-branches at once. If values cannot be predicted swiftly enough to break these data dependencies, value prediction will not be useful.



a. Loads

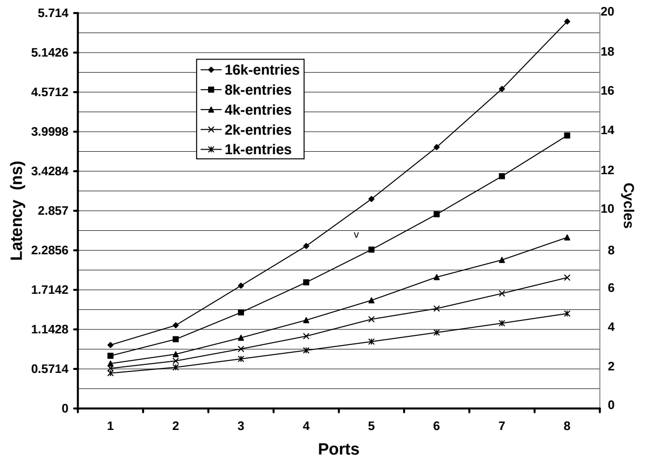


b. Integer Operations

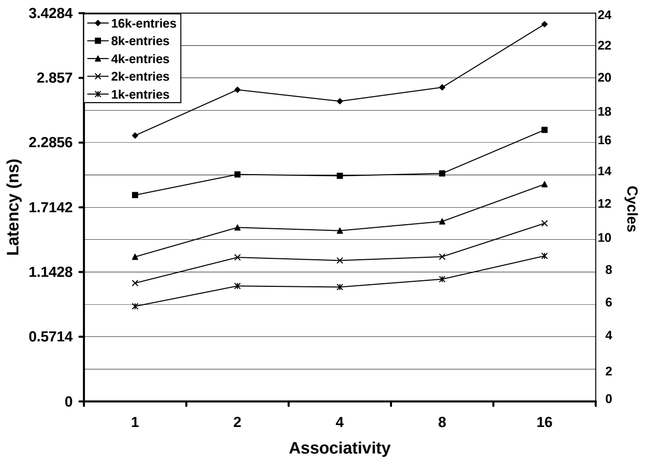
Figure 1: Distribution of Issue to Consume-Request Distances Measured in Clock Cycles
The benchmarks are the SPEC2000 integer benchmarks described in Section 3. The processor configuration for this graph is our baseline microarchitecture detailed in Section 3.1.

Table Access Latency: The push toward high frequencies in microprocessors creates a wire-delay constrained processor, causing a nontrivial latency (in clock cycles) for large centralized structures [1, 20], such as a value predictor. The latency reduces the impact of value prediction, which requires quick resolution of data dependencies to be effective.

The graphs in Figure 2 illustrate the access times for value predictors of several sizes versus the number of ports and associativity. The smallest structure in Figure 2.a, a 1k-entry predictor with one port, has a two cycle access latency. As the table size increases, the impact of additional ports



a. Varying Number of Ports (direct-mapped)



b. Varying Associativity (four ports)

Figure 2: Access Latency in Nanoseconds and Cycles for 3.5 GHz Microprocessor
A next generation processor produced using a 100nm feature size technology will run at 3.5 GHz according to projections from the Semiconductor Industry Association (SIA) [25]. Each horizontal line on the graph represents one clock period (0.2857ns) at 3.5 GHz. These latencies are obtained using Cacti 2.0, a tool for optimizing cache configurations based on latency and energy, biased toward optimizing for latency [21, 31]. We use the minimum block size of eight bytes in this graph. Cacti allows two read/write ports to be modeled. We model the first two ports this way and the remainder of the reported number of ports as one read port and one write port.

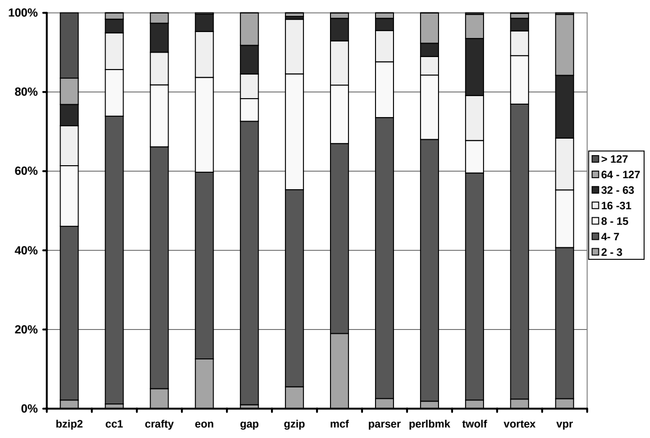
becomes increasingly severe. Figure 2.b presents latency versus associativity with the number of ports fixed at four. The associativity increase only costs one additional clock cycle for 2-8 ways since the port effects dominate the delay.

Value predictors in literature do not account for this latency. As an example, hybrid value prediction can significantly increase instruction throughput. These predictors consist of multiple types of value predictions, such as last-value prediction (LVP), stride prediction, and context prediction, being performed in parallel. The predictions are compared and one is chosen based on its prediction history. The most complex sub-predictor is the context predictor which serially accesses two tables [24, 30]. The first contains recently seen values and a value history. The second is indexed by the value history and predicts which value will come next.

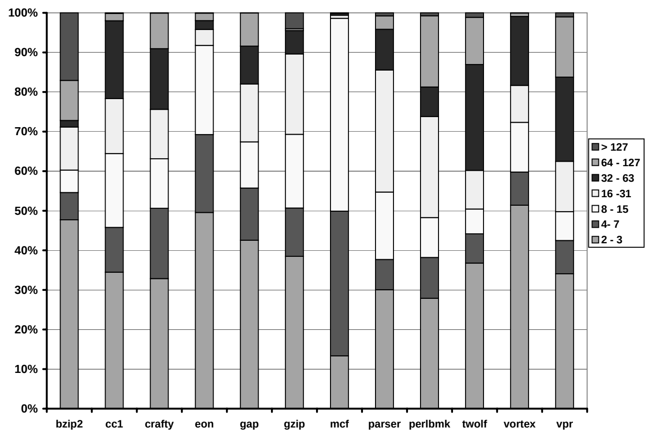
In previous literature, value predictors have been configured in many different ways. Here we discuss an aggressive configuration that is not atypical of value predictors found in literature, where each table is direct-mapped with 8192 entries and no port restrictions or latency considerations. Based on the latencies in Figure 2, accessing such a context predictor (8k-entry, 8-ports for each table) will take 28 total cycles, 14 cycles for each table. These values do not include the latency for value calculation, selection logic, and value routing.

Prediction Latency Overlaps Execution: Excessive value prediction latency is a detriment to overall instruction throughput. It reduces the effectiveness of successful predictions by prolonging the resolution of data dependencies. A lengthy latency for computing a predicted value can overlap with the out-of-order processing of instructions, allowing an instruction to produce its actual result before the predicted result is even available!

The graphs in Figure 3 depict the distribution of instruction execution latencies for result-producing instructions: loads and integer operations. For load instructions, the execution latency is measured from instruction rename to the return of a value from the memory subsystem. For integer operations, latency is from instruction rename to the completion of execution. The execution latency can be reduced or even removed by timely and successful value prediction. However, there is no point to speculating on an instruction value if the prediction latency exceeds the execution latency.



a. Loads



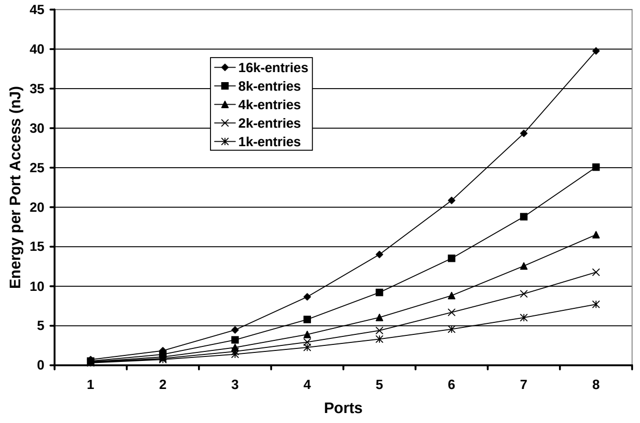
b. Integer Operations

Figure 3: Distribution of Rename to Result Times Measured in Clock Cycles
The benchmarks are the SPEC2000 integer benchmarks described in Section 3. The processor configuration for this graph is our baseline microarchitecture detailed in Section 3.1. The bars are decomposed into groupings based on the number of cycles shown in the legend.

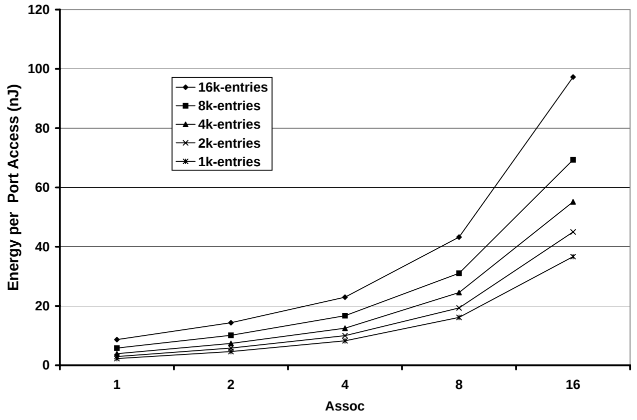
The figure reveals that value predictor latencies that extend just a few cycles past instruction rename will not be able to provide predictions quickly enough for a large percentage of instructions. A successful value prediction which is provided three cycles after an instruction is renamed will benefit 81-94% of load instructions. However, predictions provided eight cycles past rename can no longer help 41-77% of loads. Most of the load instructions fall into the 4-7 cycle category. These are the data cache hits. On the other hand, a large percentage, 13-51%, of integer operations execute within three cycles, but not many fall within the 4-7 cycle category. In both cases, approximately half of the instructions benefit from successful value predictions produced within eight cycles of the rename stage. At the other extreme, 1-23% of loads take more than 32 cycles to complete and many see benefits even with very large table access latencies. The long latency characteristics of load instructions highlights the appeal of load-only value prediction [14].

Excessive Energy Consumption: More transistors, higher clock rates, increased speculation, and wider microarchitectures all contribute to the growing problem of energy consumption. Always a concern for embedded processors, energy is now driving high-performance microprocessor design as well. Every time a large or complex piece of hardware is considered, its dynamic energy properties should be evaluated. Useful value predictors are typically composed of one or more large cache-like structures. Commonly studied table sizes range from 8KB to 160KB and beyond. In each cycle, multiple instructions must read and write to this structure to maintain high performance. In the SPEC2000 integer benchmarks, 45% to 82% of all instructions are candidates for value prediction. With this level of activity and the complexity of value predictors, an understanding of the energy requirements is essential.

Figure 4 underscores the impact that ports and associativity have on energy. While increasing either factor is expensive in terms of energy, note that the scale for associativity (number of ports fixed at four) is more than double that of the ports graph. These graphs solidify the notion that steps to increase performance can cost designers in energy consumption.



a. Varying Number of Ports (direct-mapped)



b. Varying Associativity (four ports)

Figure 4: Energy Per Port Access for Potential Predictor Sizes in 100nm Technology
The energy per port access values are obtained using Cacti 2.0. We use the model's minimum block size of eight bytes in this graph. Cacti supports a minimum of one read/write port and a maximum of two to be modeled. We model the first two ports this way and the remainder as one read port and one write port.

A value predictor may not dominate overall dynamic energy consumption, but it will contribute non-trivially. Since a value predictor is read and updated by a large percentage of instructions, it consumes more total energy than similar sized instruction and data caches. Using measured port access counts from simulation and the modeled port access energy, we find that a four-ported hybrid predictor with three 8k-entry, direct-mapped tables consumes 14 times more energy than all of the on-chip caches combined! Although on-chip caches store a lot of data, the energy consumed is not drastic since the number of ports is typically limited. In addition, the cache port accesses are much less frequent than value predictor port accesses. (More details are provided in Section 4.3).

Contributions: Value prediction is often studied in ideal and theoretical configurations to determine its potential and usefulness. In this work, we study how value prediction will perform in a wide-issue, high-frequency environment with realistic constraints, such as limited read and write ports, restricted table size, and realistic access latencies. These constraints are applied to known value predictors and proposed derivatives to understand their impact on performance. We look at the implications of performing value prediction both at the front-end and back-end of the processor pipeline. Based on our observations, we propose a latency and energy aware (LEA) form of value prediction to alleviate value prediction complexity and energy consumption while maintaining high performance. In addition to the analysis in this section, the contributions of this work include:

1. Studying the sensitivity of value prediction speedup to access latencies and port considerations. The performance gained from increasing the number of ports is tempered by the resulting increase in latency.

2. Proposing a latency and energy aware (LEA) form of value prediction. LEA prediction yields high performance, low complexity value prediction. The centralized PC-indexed value prediction tables are eliminated in LEA prediction in favor of a trace-based reads from a specialized cache (as in Decoupled prediction) and trace-based updates of buffered prediction traces.

3. Comparing performance of different high-performance methods of value prediction in the presence of latency considerations. We compare the LEA predictor to traditional At-Fetch value predictors, Post-Decode value predictors, a Post-Decode load-only value predictor, and a Decoupled value predictor [11]. The Decoupled predictor provides the best performance, achieving 10.2% speedup versus the base, while the load-only value predictor provides the least speedup, 4.5%.

4. Extending the analysis to account for the energy for a variety of value predictors. The load-only value predictor has the lowest energy requirements, consuming less than 10% the energy of the other PC-indexed predictors and less than 50% the energy of LEA predictors. LEA value prediction reduces the dynamic access energy by 58%-95% versus the complete (i.e. load and interger values) value predictors, while achieving 8.6% speedup versus our base architecture with no value prediction.

5. Evaluating read port filtering options to improve performance and reduce energy consumption

of the traditional At-Fetch value predictor. We look at several filtering techniques that improve the performance with At-Fetch value prediction by 3.2% while reducing the dynamic energy consumption by 18.3%.

The remainder of this paper is organized as follows. Section 2 explains the methods of value predictions: At-Fetch, Post-Decode, Decoupled, and LEA. Section 3 describes our evaluation methodology. Section 4 compares the four scenarios with respect to performance and energy. Section 5 discusses previous value predictor work and fine-tuning. Section 6 summarizes and concludes the paper.

2 Value Predictor Placement and Implementation Choices

We examine multiple points at which an instruction is value predicted. Two basic methods are seen in literature, At-Fetch front-end prediction and Decoupled back-end prediction. Each stage has advantages and drawbacks which are discussed below. To address the drawbacks, we study Post-Decode prediction as a front-end alternative and Latency and Energy Aware (LEA) prediction as a back-end option.

2.1 At-Fetch Value Prediction

Performing value prediction at instruction fetch is a commonly assumed implementation [4, 9, 13, 22]. In a typical processor, an instruction address can be sent to the fetch logic each cycle. This same fetch address is used to access the value predictor. Based on the fetch PC, the value predictor generates predictions for all of the instructions being fetched in that cycle.

This imposes two restrictions. In a processor that fetches past branches in a single cycle (such as trace cache processors), the instruction PCs can be non-contiguous. Determining the PC for each fetched instruction requires more information than is typically available at fetch. Solutions that may work when fetching up to the first branch, such as banking, no longer work well [9]. The second problem is the lack of instruction type information. During fetch, the instructions are indistinguishable, so value prediction resources are being consumed by instructions that are not eligible for predictions (branches, stores, floating point). This proves costly for a port-constrained predictor.

The advantage of At-Fetch prediction is compelling. There is no need for a predicted value until the instruction has been decoded and renamed. Therefore, some or all of the value predictor's table access latency is hidden by the instruction fetch latency and decode stages.

2.2 Post-Decode Value Prediction

An alternative to predicting at fetch time is to predict after the instruction is decoded. After instruction decode there is more information available about the instruction. The PCs for non-contiguous instructions beyond the first branch are now known, allowing more instructions to access the value prediction hardware. Using the instruction type information obtained at decode, more judicious read port access arbitration can take place, limiting access only to instructions that generate results. Access to this post-decode information is commonly assumed for At-Fetch predictors.

The disadvantage of Post-Decode prediction is that the value predictor access latency can not be hidden by earlier stages of the pipeline. The tables are accessed when the predicted value is ideally desired. Every extra cycle means a delay in breaking a possible data dependency. If there is any delay in obtaining a value prediction, the corresponding instruction will have already entered the instruction window.

2.3 Decoupled Value Prediction

Predicting at fetch or after decode involves computing the predicted value just before an instruction needs it. Instead, prediction computation for an instruction can be done immediately after a previous instance of the instruction retires. In Decoupled value prediction, an instruction retires and updates the value predictor table as in traditional value prediction. Then the retired instruction accesses the value predictor again right away and computes a new prediction. This predicted value is stored in a Prediction Value Cache (PVC) and can be used by the next instance of the instruction. Two versions of the Decoupled value predictor have been presented, one for machines with trace caches [11] and one for machines with no trace cache [12]. The trace cache strategy is used in this paper since it first perfectly within the wide-issue environment which we are studying. The following summary applies to both strategies.

The fill unit [26] stores the predicted value and some additional bits for each instruction in the PVC. This cache is indexed just like the instruction cache (either a traditional instruction cache or a trace cache) and there is a one-to-one mapping of instruction to predicted values in the PVC. Therefore, based on the fetch PC, one prediction is available for each instruction in the cache line. The cache and PVC have similar sizes and latencies, so the predictions arrive in parallel with the fetched instructions. This eliminates both the access latency problem and the port arbitration problems.

From a complexity and energy standpoint, Decoupled value prediction offers little help. It maintains the traditional centralized PC-indexed value predictor table and structure. At retire-time this table must be both read to perform new value predictions and updated by retiring instructions. The number of read accesses decreases since wrong-path instructions (from mispredicted branches) do not ever access the value prediction tables. The additional PVC also requires area and consumes energy with its own reads and updates, although they occur with less frequency than value predictor table updates.

The primary performance weakness of this approach is the dependency on the cache performance. Predictions are lost whenever a cache line is kicked out. Any instructions fetched from outside the cache receive no prediction. Another problem is staleness. Gathering a prediction for an instruction immediately after updating will produce a different result than waiting for the instruction to be fetched again. This is especially true if multiple instances of the instruction are present in the instruction window simultaneously due to a tight loop.

2.4 LEA Value Prediction

In this section we present a latency and energy aware (LEA) value prediction technique to address the energy and complexity concerns of Decoupled value prediction. The novelty of LEA prediction is that it does away with the centralized PC-indexed value prediction table. Instead, predicted values are always kept in trace form and trace-based updates are performed in the fill unit. This is made possible by including stride information in the PVC along with the predicted value.

As shown in Figure 5, value predictions are fetched in trace format from the PVC and fed to the processor core as in Decoupled prediction. This method of value prediction reduces complexity and energy versus Decoupled value prediction by eliminating the centralized, multi-ported PC-indexed tables and replacing them with value prediction updates on buffered prediction traces. This requires a separate Prediction Trace Queue (PTQ) in the fill unit to buffer the prediction traces. This queue is more energy efficient compared to value prediction tables since it is direct-mapped, requires only one read and one write port, needs fewer entries (since multiple predictions are stored in one entry), and is accessed fewer times.

The prediction traces persist in the PTQ until the related instructions retire. While the fill unit is creating new traces from the retired instructions, the fill unit value prediction logic compares the final values to the predicted values from the oldest trace in the PTQ [1]. New predictions are then combined by the fill unit logic into a prediction trace which is stored in the PVC.

We analyze two types of LEA prediction. One version, referred to simply as `LEA`, uses basic stride prediction which requires only the value, a 16-bit stride, and two bits for confidence. The other version does matched (also know as two-delta) stride prediction which requires another 16-bit

[1]Wrong-path instructions exist in the PTQ, but it is relatively simple to recognize this condition and ignore or discard those predictions.

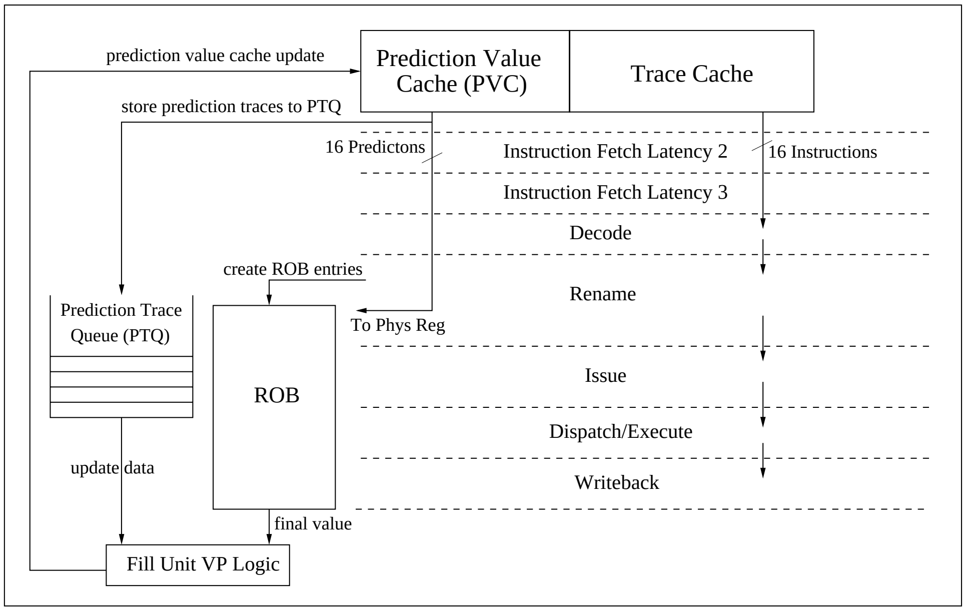

prediction value cache update
Prediction Value Cache (PVC)
Trace Cache
store prediction traces to PTQ
16 Predictons
Instruction Fetch Latency 2
16 Instructions
Instruction Fetch Latency 3
Decode
create ROB entries
Rename
Prediction Trace Queue (PTQ)
To Phys Reg
Issue
ROB
Dispatch/Execute
update data
Writeback
final value
Fill Unit VP Logic

Figure 5: Latency and Energy Aware (LEA) Value Prediction
Up to 16 instructions and corresponding predictions are fetched together from the trace cache and PVC respectively. The instruction trace passes through the pipeline as usual. The trace of predictions advance in a similar manner, and are placed in the ROB during instruction rename. The prediction trace is then stored in the PTQ until it is needed by the Fill Unit at instruction retirement.

stride. We call this `LEA+`.

LEA prediction is restricted to performing value prediction based on one-level algorithms, such as stride prediction. Context prediction, and thus hybrid prediction, are no longer feasible without requiring an extra table and severely increasing the PVC and PTQ sizes. Also, the PVC experiences the redundancy phenomenon seen in trace caches [17, 19]. Prediction values for a particular instruction can exist in multiple prediction traces if the instruction is found on multiple paths. However, trace-like storage is often an advantage due to correlation and context effects.

3 Methodology

In this section, we describe the benchmarks and simulation environment used to study value prediction trade-offs. We examine the integer benchmarks from the SPEC CPU2000 suite [27]. The SPEC programs are dynamically linked and compiled using the Sun CC Compiler with the `-fast -xO5` optimizations. The benchmarks and their respective inputs are presented in Table 1. All benchmarks are run for 100 million instructions after skipping the first 100 million instructions.

Table 1: SPECINT2000 Benchmarks

Benchmark	Inputs
bzip2	input.random 2
crafty	crafty.in
eon	chair.control.kajiya chair.camera chair.surfaces chair.kajiya.ppm ppm pixels_out.kajiya
gap	-l ./ -q -m 64M test.in
gcc	cccp.i -o cccp.s
gzip	input.compressed 2
mcf	inp.in
parser	2.1.dict -batch test.in
perlbmk	-I. -I./lib splitmail.pl 1 5 19 18 150
twolf	test
vortex	bendian.raw
vpr	net.in arch.in place.out dum.out -nodisp -place_only -init_t 5 -exit_t 0.005 -alpha_t 0.9412 -inner_num 2

3.1 Baseline Microarchitecture

To perform the simulations, we use a functional, program-driven, cycle-accurate simulator. Given a SPARC executable as input, the front-end functional execution is performed by the Sun tool Shade [5]. The simulator models aggressive instruction-level parallelism techniques, as well as all resource contentions and speculative execution. The basic pipeline consists of eight stages: three stages of fetch plus decode/merge, rename, issue, execute, and writeback. Memory operations require additional pipeline stages, including TLB access and cache access. We assume the same clock frequency in all simulations (3.5 GHz), and adjust value predictor latencies based on this fixed time period. The parameters for the simulated base microarchitecture can be found in Table 2.

Table 2: Baseline Microarchitecture Configuration

Data memory	
· L1 Data Cache:	4-way, 32KB, 2-cycle access
· L2 Unified cache:	4-way, 1MB, 10 cycles
· Non-blocking	12 MSHRs and 2 ports
· D-TLB	512-entry, 4-way, 1-cycle hit, 30-cycle miss
· Store buffer:	32-entry w/load forwarding
· Load queue:	32-entry, no speculative disambiguation
· Main Memory	Infinite, 75 cycles

Fetch Engine	
· Trace cache:	4-way, 1K entry, 3-cycle access partial matching, no path assoc.
· L1 Instr cache:	4-way, 4KB, 1-cycle access one basic block per access
· Branch Predictor:	16k entry gshare/bimodal hybrid predictor
· Branch target buffer	512 entries, 4-way

Execution Core			
· Functional unit	#	Exec. lat.	Issue lat.
Load/store	6	1 cycle	1 cycle
Simple Integer	8	1	1
Int. Mul/Div	2	3/20	1/19
Simple FP	4	3	1
FP Mul/Div/Sqrt	1	3/12/24	1/12/24
Branch	4	1	1
· Data Forwarding Latency: 1 cycle			
· Register File Latency: 2 cycle			
· 128-entry ROB			
· 8 reservation station entries per func. unit			
· Fetch width: 16			
· Decode width: 16			
· Issue width: 16			
· Execute width: 16			
· Retire width: 16			

The trace cache and value predictor latencies are modeled with Cacti 2.0 [21]. The trace cache access latency is noticeably lower because it is modeled with just one read/write port. The configurations of the remaining cache structures are intentionally aggressive and chosen to place a little extra pressure on the instruction window and execution resources. In our configuration, Cacti 2.0 projects a four cycle latency for our data cache configuration. Applying this latency would reduce overall processor performance, but increase the relative benefit of all forms of value prediction. However, we simulate a more aggressive two-cycle latency so that the load latency effect is reduced.

Similarly, the gshare branch predictor [15] and BTB are aggressively modeled to maintain a steady instruction fetch rate and place a little extra pressure on the instruction window. Similarly, the trace cache multiple branch predictor is optimistically modeled by accessing the gshare predictor and BTB multiple times per cycle. Multiple branch prediction is a separate area of research, but we believe our method of branch prediction modeling simulates an achievable target for future multiple branch predictors.

3.2 Simulated Value Predictors

The simulated value predictors are shown in Table 3. The `Unconstrained` configuration is an aggressive but unrealistic value predictor with no port restrictions and zero latency presented for comparison purposes. For all predictors, each table is direct-mapped, tagged [2], and updated by every result-producing instruction. When a value mis-speculation is encountered, the microarchitecture reissues all instructions younger than the mispredicted instruction.

All value predictors are assumed to be fully pipelined and capable of serving new requests every cycle. Since the clock frequency is fixed, we alter the access latency when we change value predictor configurations. The modeled latency from Cacti is divided by the clock period to produce the latency. This is an optimistic assumption considering the required pipelining overhead.

Table 3: Analyzed Value Predictors

Predictor	LVP Entries	Stride Entries	Context Entries	Ports	Total Prediction Lat.	Unhidden Lat.
Unconstrained	8192	8192	8192	16	0 cycles	0 cycles
At-Fetch Hybrid	8192	8192	1024	4	8 cycles	4 cycles
At-Fetch Stride	N/A	4096	N/A	4	5 cycle	1 cycle
Post-Decode Hybrid	8192	8192	1024	4	8 cycles	8 cycles
Post-Decode Stride	N/A	4096	N/A	4	5 cycles	5 cycles
Post-Decode Stride Load	N/A	4096	N/A	2	3 cycles	3 cycles
Decoupled	8192	8192	8192	4	3 cycles	0 cycles
LEA/LEA+	N/A	N/A	N/A	N/A	3 cycles	0 cycles

`Ports` refers to the available ports at read or write, modeled as two read/write ports and then matching read-write pairs beyond that. `Total Prediction Lat.` is the latency in clock cycles from value prediction request until a value is produced. `Unhidden Lat.` is the number of total prediction latency cycles that extend past rename.

[2] Using tags trades energy savings for noticeable performance, even in direct-mapped predictors. This is a difficult trade-off and we ultimately use tags so that the Cacti (a cache simulator) results will be more faithful to our simulated predictors.

The choice of table entries for the context predictor is based on the observation that stride prediction is the main contributor to hybrid performance. For hybrid configurations we picked a high-performing stride predictor configuration and used its latency to fix the sizes of the other sub-predictors. In the case of a 4k-entry stride predictor, there is no reasonable two-level configuration that can be accessed in parallel.

The `Post-Decode Stride Load` predictor only speculates on load instruction values. The other presented value predictors are *complete value predictors*, predicting values for all result-producing operations. Figure 1 shows that load instructions produce values that are consumed more quickly than integer instructions, and Figure 3 shows that load values are obtained more slowly than integer computation values. This confirms a previous notion that load instructions are a more appealing target for value prediction. Therefore, we include a stride-based load-only value predictor. The configuration presented in Table 3 is the best performing within reasonable energy considerations . Note that unlike the complete value predictors, only two ports are needed.

The unhidden latency for the Post-Decode predictors equals the total prediction latency. The unhidden latency for the At-Fetch predictors is the total prediction latency minus the four fetch and decode pipeline stages. Table access latency only accounts for obtaining data from predictor entries. The process of calculating and choosing a value and its prediction status is lumped into one cycle for hybrid predictors and ignored for stride predictors.

We apply update latencies upon completion of an instruction. Modifying this latency does not affect performance significantly. An update is a read-modify-write operation so this latency is chosen to be at least twice the read latency. We assume unlimited write bandwidth at update time. For the Decoupled predictor, there is no latency beyond decode to worry about. The Decoupled predictor is tolerant of very large update latencies without losing performance. Therefore we model the Decoupled value predictor with unlimited update-time read and write bandwidth per cycle. In practice, the predictor would not be updated and read every cycle. When it is accessed, the average number of instructions that wish to access the value predictor is about four. A strategy such as the update queues suggested in [11] can effectively provide infinite bandwidth with the ample latency available at update. The LEA predictors have a zero cycle update latency since it has no PC-indexed value prediction table to access at update.

The two-level value predictor used in hybrid prediction is similar to the one discussed by Wang and Franklin [30]. It has internal confidence counters to reflect the likelihood that a value is the next one in a pattern. The second level of the two-level predictor uses a saturating counter associated with each value which saturates at 12 and has a prediction threshold of 9. The counter is incremented by three when the value is the next one in the pattern, and decremented by one if it is not.

The stride predictors studied use matched (two-delta) stride prediction [6]. The stride predictor uses a two-bit saturating counter for its internal prediction confidence mechanism. It works in a very traditional manner. It is incremented by one when the prediction is correct and decrement by one when it is incorrect. When the counter value is greater than two, then an actual value speculation can occur.

Each sub-predictor in the hybrid predictors tracks its prediction confidence with a four-bit saturating counter. The confidence is incremented by one when the sub-predictor is correct and decremented by three when it is incorrect. The hybrid predictor uses the prediction from the sub-predictor with the highest confidence, as long as that value is greater than the threshold of 12. For indicated hybrid predictor simulations, we simulate a perfect confidence mechanism. In this case, if any of the sub-predictors predict the correct value, then this prediction is chosen. Even with a perfect confidence mechanism, the value predictor can mispredict if none of the sub-predictors predict the proper value.

4 Analysis

We first analyze the sensitivity of value prediction to latency, access ports, and predictor size. Understanding these effects, we perform a comparison of the different value prediction strategies. After analyzing these predictors strictly from a performance point of view, we study the dynamic energy implications of each configuration. Finally, we present energy-saving and performance-enhancing techniques for traditional At-Fetch and Post-Decode value predictors.

For aesthetic reasons, in the speedup figures, the geometric mean of *all 12* SPECint2000 benchmarks is presented along with select individual benchmarks. These selected benchmarks represent different performance levels attainable with value prediction. We chose three programs that show the highest performance speedup from value prediction (`perlbmk`, `bzip2` and `gap`), two that show reasonable speedup (`crafty` and `parser`), and three that show low speedup (`mcf`, `cc1` and `eon`). The other four benchmarks do not show visually different trends from the selected benchmarks but are included in the average or geometric mean.

4.1 Effects of Table Access Latency on Performance

Figure 6 quantifies the impact of changing value predictor latency on performance using a version of the `Unconstrained` predictor that has non-zero latencies. The latency (`lat`) is the unhidden latency, i.e. the number of cycles between when an instruction is renamed and when its predicted value becomes available. As a reference, the assumption in previous work is that this latency is zero, and the hybrid predictor with large 8k-entry tables described in Section 1 experiences a total table access latency of 28 cycles, 24 of which are seen after rename.

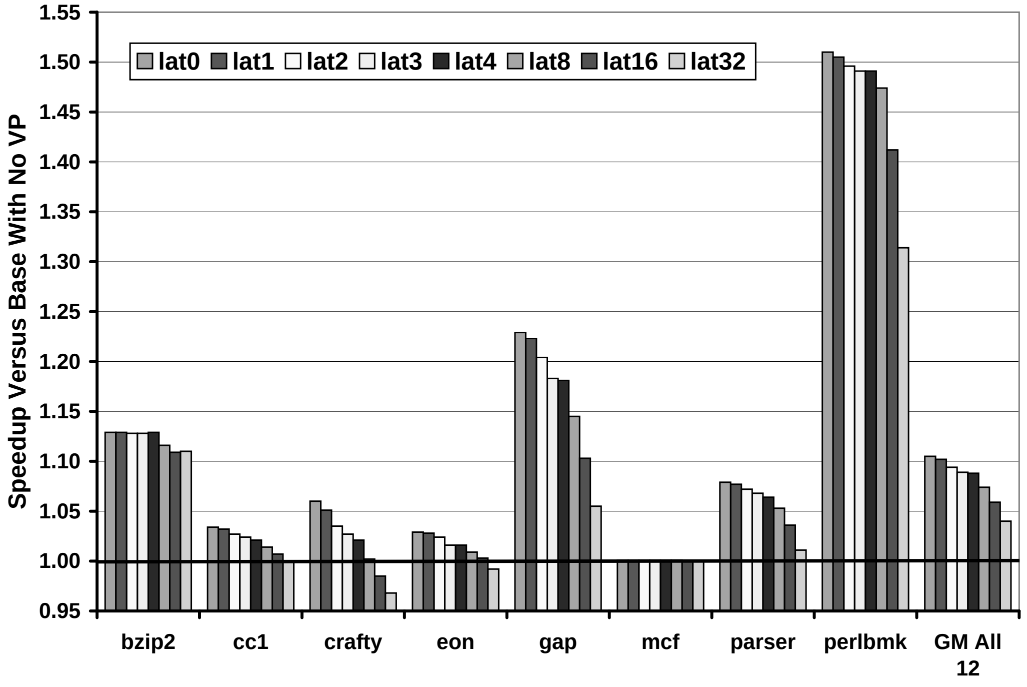
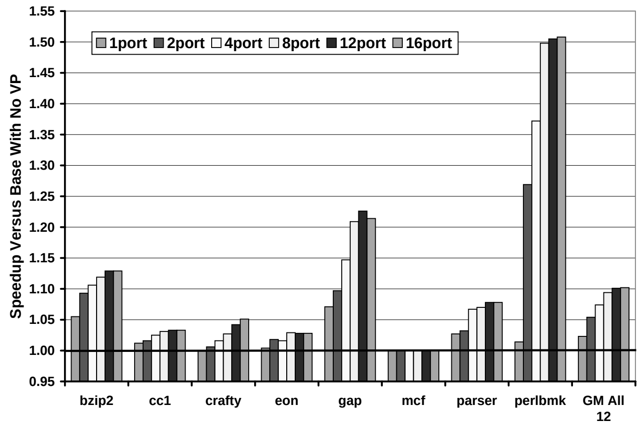


Figure 6: Effect on Performances of Unhidden Table Access Latency
`lat` is the number of cycles after an instruction is decode that its predicted value becomes available.

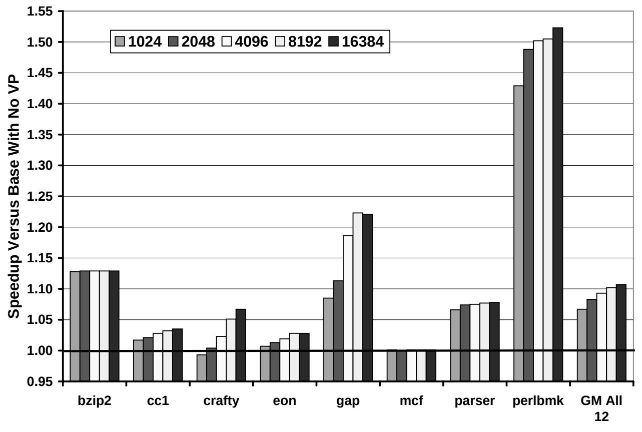
The figure shows that unhidden latency can significantly affect performance. One extra cycle of value predictor latency reduces overall performance, but by less than 0.5% for the whole benchmark suite. Applying four cycles of unhidden latency reduces the absolute speedup by 1.7%, a significant portion of the original 10.5% speedup. As the unhidden latency increases, a larger percentage of instructions are able to compute a result for themselves and the benefits of successful value prediction deteriorate. However, it is interesting to note that even with 32 cycles of unhidden latency, there is still potential for achieving 4.0% speedup with value prediction. Although few instructions take 32 cycles to complete execution, the impact of successfully predicting their results still noticeably helps ILP, even after many cycles of waiting for the predicted value.

Factors that lead to significant value prediction table access latency include the number of ports and the number of entries in the tables. Access ports are isolated in Figure 7.a using the `Unconstrained` configuration. Port arbitration is done on a first-come, first-serve basis. Therefore, decreasing the number of ports reduces the number of instructions that have the opportunity to access their prediction.

Figure 7.a illustrates that reducing read ports to 12 in a 16-wide environment barely affects performance. Having eight read ports limits value prediction coverage, but still provides reasonable performance, 1.2% below the 16-port configuration on average. However, reducing the number of read ports to four cuts the average speedup from 10.2% to 7.4% versus a 16-port configuration. With an average trace length of around 11 instructions, a significant number of predictable instructions will never get a chance to access the value predictor.



a. Ports



b. Size

Figure 7: Effect on Performance of Restricted Ports and Table Entries
Ports are expressed in the number of read and write ports. This is modeled in Cacti as two read/write ports and read and write port pairs after two.

In Figure 7.b, the number of table entries is altered for a hybrid predictor. The number of entries shown in the legend apply to all of the tables in the hybrid predictor. The largest configuration is double the size of the `Unconstrained` predictor. Each subsequent configuration is obtained by reducing the table by half, ending with 1024 entries. On average, the 8k-entry hybrid predictor provides very similar performance to the 16k-entry predictor. With 1k-entry tables, it is still possible to reach 62% of the 16k-entry predictor speedup. However, to maintain high-performance 4k-entry and 8k-entry tables are attractive.

4.2 Comparing Prediction Strategies

Previous analysis shows that tables with eight ports lead to high access times and energy consumption, while two ports results in a significant performance hit. Tables with less than 4k entries are too small for high performance, but those with more than 8k entries provide little benefit given the energy and latency penalty. This leads us to focus on two port/size configurations for the complete value predictors, 4k-entry tables with four ports and 8k-entry tables with four ports (as discussed in Section 3.2, Table 3).

A two-port, 4096-entry, post-decode, stride load-only value predictor is also evaluated. This predictor only targets load instructions, thus the number of ports necessary for high performance is fewer. We found that post-decode is a better performance and energy strategy for load-only value prediction. The selective port usage of post-decode prediction outweighs the benefits of the reduced latency of at-fetch prediction.

Figure 8 compares the overall performance of each chosen value prediction strategy and latency-constrained configuration. In this section, the Unconstrained, At-Fetch and Post-Decode hybrid predictors use perfect confidence for choosing among the sub-predictors. The two versions of LEA prediction are analyzed. One performs matched stride prediction (`LEA+`) while the other performs simple stride prediction (`LEA`).

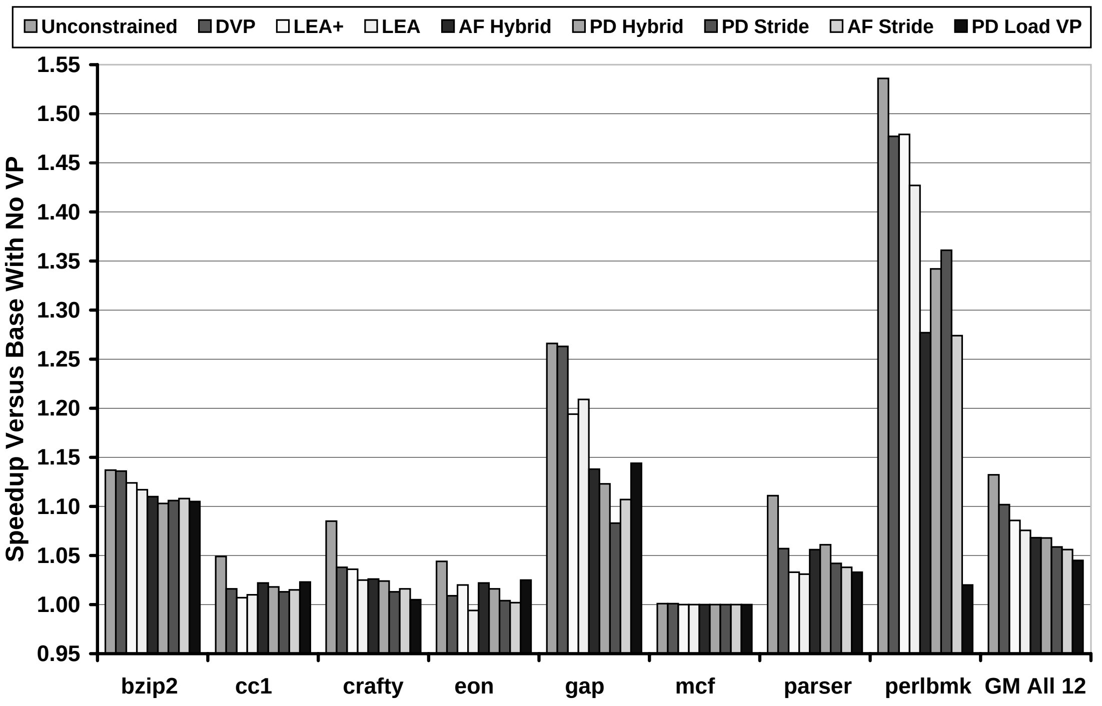


Figure 8: Performance Comparison of Prediction Strategies
`DVP` is Decoupled value prediction. `AF` is At-Fetch value prediction. `PD` is Post-Decode value prediction. `PD Load VP` is the the post-decode load-only value predictor.

On average, Decoupled value prediction has the best performance, achieving 10.2% speedup over the base model. Expectedly, it does not reach the performance of the Unconstrained predictor, but does achieve 77% of its speedup. When using the hybrid choosing mechanism described in Section 3.2 instead of perfect confidence, the speedup is still better than any other option (9.3% speedup; not shown in graphs). The two versions of LEA achieve the next best performance, with LEA+ (8.6% speedup) expectedly outperforming LEA (7.5% speedup). The traditional At-Fetch hybrid predictor produces a 6.8% speedup but falls far short of Unconstrained. This is notable since value predictors assumed to be At-Fetch predictors are often modeled more closely to the Unconstrained predictor.

Interestingly, the Post-Decode hybrid predictor matches the performance of the At-Fetch hybrid despite incurring a larger unhidden latency. Post-Decode predictors perform more successful predictions as a result of better read port utilization. A Post-Decode predictor benefits more when using smaller table sizes than an At-Fetch predictor since all of the latency reduction is unhidden latency reduction.

The load-only value predictor performs the worst, achieving 4.5% speedup over the base. Even under aggressive assumptions (`Unconstrained` configuration performing load-only prediction), only 6.2% speedup is possible when predicting just load instruction values. Even though there is less urgency to predict integer operations and the prediction latency is less than that of load instructions, the importance of speculating on these values is important for achieving large speedups in wide-issue microprocessors.

The performance superiority of the Decoupled and LEA predictors is not surprising since the latency beyond decode is intrinsically zero. In addition, the prediction read bandwidth is effectively unconstrained because of the trace prediction format. These performance results highlight the advantage of performing prediction at retire time. The advantage of Decoupled over LEA is due to the use of a hybrid prediction versus just matched stride prediction. The staleness concerns mentioned earlier do not manifest in these simulations. All prediction strategies face this problem to some degree because of the large instruction window and retire-time updates. Speculative updating of the value predictor (which is complex and high energy) is required for the At-Fetch and Post-Decode strategies to realize an advantage over predictors that predict at instruction retire.

4.3 Energy Analysis

Figure 9 reports relevant read and write frequencies for a predictor similar to the `Unconstrained` value predictor, but restricted to four ports. The first column is the number of port accesses that take place during execution due to value predictor reads (`VP Read`). Only result-producing instructions access the predictor since it is a Post-Decode predictor. The next column, `VP Ld Read`, presents the number of read accesses during load-only value prediction. The next two columns, `VP Update` and `VP Ld Update`,, are the number of port accesses made at instruction retire due to updates to the value predictor for complete and load-only prediction, respectively.

The `TR Read` column is the number of trace cache hits. Since the PVC is synchronized with the trace cache in this work, this number is equivalent to the number of port accesses due to PVC reads for Decoupled and LEA predictors. (The `TR Read` value for `mcf` is less than 100,000 and therefore looks like zero on this graph.) The final column represents trace cache builds, `TR Build`. Retired

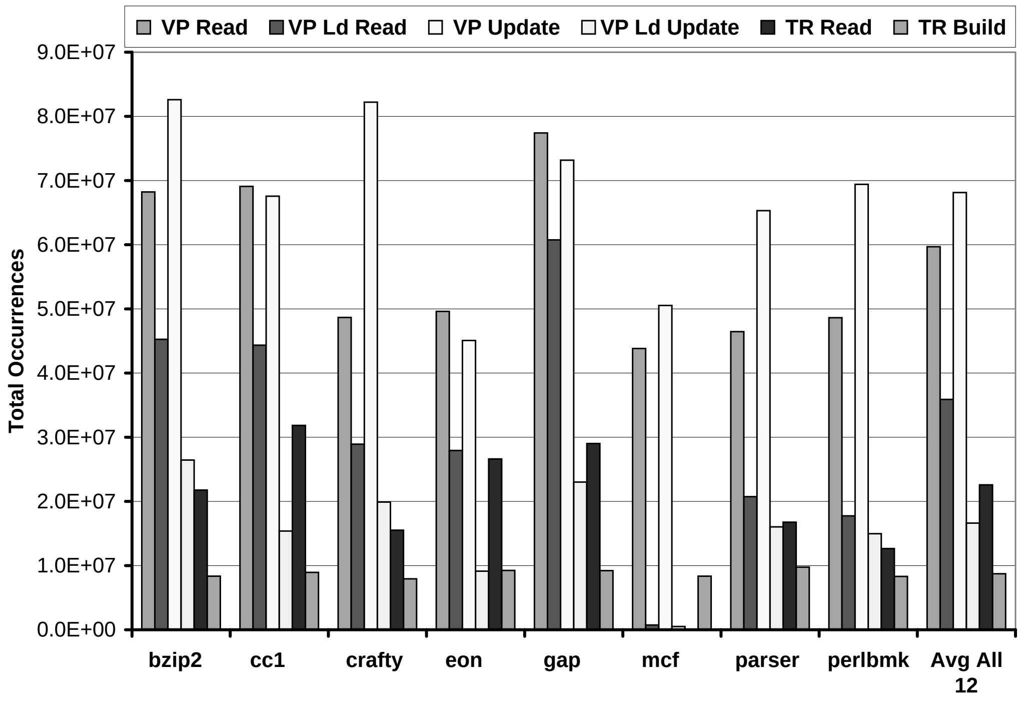


Figure 9: Table Port Access Frequencies
The predictor used is a four-ported hybrid Post-Decode predictor with 8k-entries per table and a 0 total access latency. `VP Ld Read` and `VP Ld Update` are performed on a load-only predictor.

instructions are coalesced to make a trace and then selectively written to the trace cache by the fill unit. While a trace of instructions is frequently similar to a currently stored trace, traces of values almost always change so the PVC is written on each trace build.

`VP Update` is greater than `VP Read` when there are a lot of predictable instructions that do not get access to read ports. since these instructions still perform updates. It is assumed that all instructions will be able to perform an update. This is possible because the write bandwidth is very low and update buffers are practical at completion-time [11, 23]. `VP Read` is greater than `VP Update` when a large number of wrong-path instructions access the read ports. These instructions do not ever perform updates. `VP Ld Read` and `VP Ld Update` are performed on a load-only predictor. Therefore, when compared to complete value prediction (`VP Read` and `VP Update`) the percentage of read accesses that are loads does not equal the percentage of updates due to loads. Once again, this is because a limited number of instructions can read the predictors while all instructions are able to update the predictors.

Table 4 outlines all of the value prediction hardware used in the evaluated predictors. For instance, the At-Fetch hybrid predictor from the previous sections consists of the two 1k-entry levels of the context sub-predictor, the 8k-entry stride sub-predictor, and the 8k-entry LVP sub-predictor. Similarly, the LEA predictor requires the 1k-entry PVC and a 32-entry PTQ. Three different PVC structures are presented, allowing for the differing sizes of per-instruction data that are stored for the Decoupled, LEA, and LEA+ predictors. Note that the load-only 4k-entry table has two ports, while all other value prediction tables have four.

To determine the total energy for each prediction strategy, we use the relevant measurements in Figure 9 for each predictor and then multiply those totals by the corresponding energy per port access from Table 4. The exception is the number of value predictor reads for the load-only value predictor, which uses a two-ported table instead of the four-ported table from Figure 9. We assume that the energy to perform a write is equivalent to the energy to perform a read. The equations used for determining the overall dynamic energy of our value prediction strategies are presented below Table 4.

Table 4: Energy Parameters for Value Predictor Structures

Structure	Notation	Entries	Block Size	Assoc	Ports	Energy Per Port Access(nJ)
1k-entry Context L1 Table	1kCtxtL1_E	1024	16B	1	4	2.7108
1k-entry Context L2 Table	1kCtxtL2_E	1024	8B	1	4	2.2605
4k-entry Stride/LVP Table	4kTable_E	4096	8B	1	4	3.8961
4k-entry Stride Table Ld VP	4kTableLdVP_E	4096	8B	1	2	1.0604 nJ
8k-entry Stride/LVP Table	8kTable_E	8192	8B	1	4	5.7957
Decoupled Prediction Value Cache	DVPPVC_E	1024	64B	2	1/1	2.4213
LEA Prediction Value Cache	LEAPVC_E	1024	96B	2	1/1	2.6194
LEA+ Prediction Value Cache	LEA+PVC_E	1024	128B	2	1/1	5.1731
LEA Prediction Trace Queue	LEAPTQ_E	32	96B	1	1/1	0.5435
LEA+ Prediction Trace Queue	LEA+PTQ_E	32	128B	1	1/1	0.5847

The energy per port access is for a read access and is calculated using Cacti 2.0. The `1/1` value for `Ports` refers to one read/write port and one read port. The `Ports` for the value prediction tables are modeled as two read/write ports plus two read and write port pairs. Eight bytes is the minimum allowable block size in the Cacti 2.0 model.

Equations used to determine total energy for each predictor strategy:
[1] At-Fetch Hybrid_E = (VP Read * hybrid_E) + (VP Update * hybrid_E)
[2] At-Fetch Stride_E = (VP Read * 4kTable_E) + (VP Update * 4kTable_E)
[3] Decoupled Hybrid_E = (VP Update * hybrid_E) + (TR Read * DVPPVC_E) + (TR Build * DVPPVC_E)
[4] Decoupled Gated_E = (VP Update * 4kTable_E) + (TR Read * DVPPVC_E) + (TR Build * DVPPVC_E)
[5] Post-Decode Stride Ld VP_E = (VP Ld Read * 4kTableLdVP_E) + (VP Ld Update * 4kTableLdVP_E)
[6] LEA+_E = (TR Read * LEA+PVC_E) + (TR Build * LEA+PVC_E) + (2 * TR Read * LEA+PTQ_E)
[7] LEA_E = (TR Read * LEAPVC_E) + (TR Build * LEAPVC_E) + (2 * TR Read * LEAPTQ_E)
where hybrid_E = (8kTable_E + 8kTable_E + 1kCtxtL1_E + 1kCtxtL2_E)

For example in Equation 1, the energy consumed for the at-fetch hybrid predictor is equal to the number of value predictor read port accesses (`VP Read`) times the energy of a hybrid predictor (`hybrid`$_E$), plus the number of value predictor write port accesses (`VP Write`) times the energy of a hybrid predictor. The energy of the hybrid predictor is the energy of an 8k-entry stride predictor table (`8kTable`$_E$) plus an 8k-entry last value predictor table (`8kTable`$_E$) plus the two levels of context predictor tables (`1kCtxtL1`$_E$ and `1kCtxtL2`$_E$).

To give a relative picture of the dynamic energy consumed by value prediction, Figure 10 compares the energies of the value prediction strategies to the dynamic energy of the L1 data cache, L1 instruction cache, L2 cache, and trace cache from our baseline microarchitecture. Moreno et al. observe that a classifying hybrid predictor could allow for only one prediction table to be "turned on" during value prediction, we cover this possibility with `Decoupled Gated`. In this configuration, only the stride predictor energy is consumed and the other tables are assumed to consume no energy.

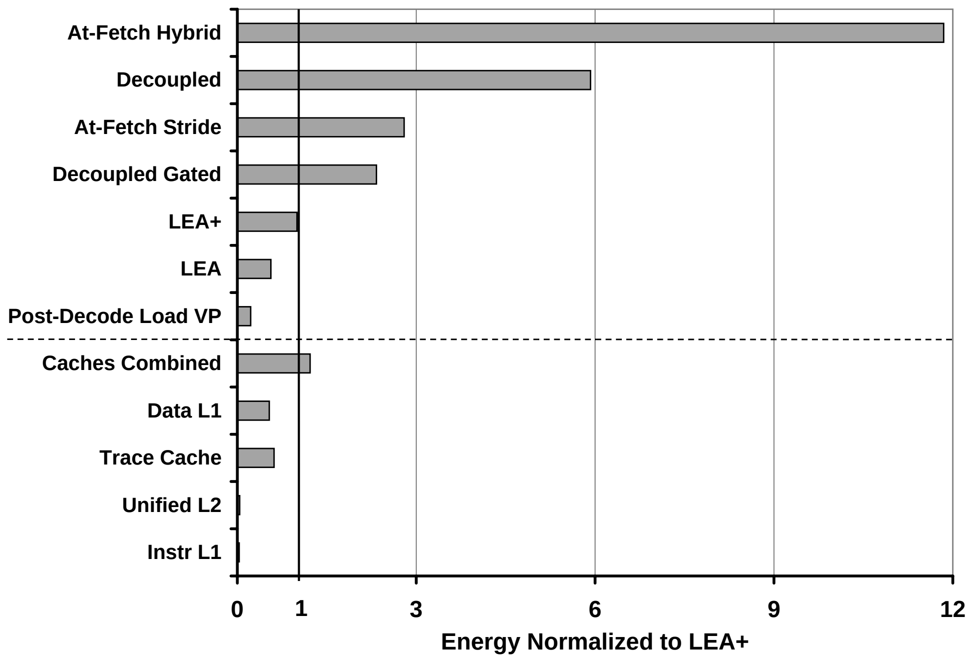


Figure 10: Energy Consumption of Value Predictors and On-chip Caches (Normalized to LEA+)
`Caches Combined` is the Data L1, Trace Cache, Instruction L1, and Unified L2 combined energy

Studying just the value predictor energy results, the hybrid predictors consume a lot of dynamic energy compared to the other configurations. The stride predictor versions of At-Fetch and Decoupled prediction do significantly better, achieving 4.1 and 2.8 times less dynamic energy. However, LEA prediction is by far the most energy-efficient method of value prediction. Trading the PC-indexed value prediction tables for a smaller, single-ported trace-accessed Prediction Trace Queue is the main source of energy reduction. LEA+ uses one-fifth the energy of the next-best non-LEA predictor. LEA with basic stride prediction has smaller entries and is even more energy efficient, reducing dynamic energy by almost one-half versus LEA+. This energy analysis highlights the primary benefits of LEA prediction.

The load-only value predictor requires the least energy consumption. This is due to a table with less energy per port access and fewer port accesses. The two-ported table provides a considerable per-access reduction in energy compared to the other four-ported PC-based tables. Limiting predictor read and update to only load instructions, reduces the number of port accesses by eliminating all integer operations. Finally, unlike complete value prediction, load-only prediction sees a reduction in energy consumption when doing post-decode prediction. Port utilization is able to decrease due to the relative infrequency of load operations.

The lower portion of Figure 10 presents the energy consumed by the on-chip caches. Amazingly, At-Fetch hybrid predictor consumes 14 times the dynamic energy as all of the on-chip caches combined. The At-Fetch stride predictor and the gated Decoupled predictor prove to be more energy friendly, but still consume 3.29 and 2.74 times the energy of the combined caches. This is a direct result of the value predictor ports being accessed much more frequently than the cache ports. Each instruction must access a port for reading and writing. However, in the trace cache, multiple instructions are retrieved with just one port access. These results assume that either the instruction cache or the trace cache data is accessed, but never both on the same fetch. Finally, only the LEA predictors have similar energy consumption to the caches. The configurations and per port access energies for the on-chip caches are presented in Table 5.

Table 5: Energy Parameters for On-Chip Caches

Structure	Entries	Block Size	Assoc	Ports	Energy Per Port Access
L1 Instruction Cache	128	32B	4	1	1.1639 nJ
L1 Data Cache	1024	32B	4	2	3.7377 nJ
L2 Unified Cache	32768	32B	4	1	8.5163 nJ
Trace Cache	1024	128B	4	1/1	5.1731 nJ

The energy per port access is for a read access and is calculated using Cacti 2.0. The `1/1` value for the Trace Cache `Ports` refers to one read/write port and one read port. The `Ports` for the other caches are modeled as read/write ports.

Determining the best value prediction scheme is often dependent on both energy and performance. One metric that merges both of these factors it the energy delay product. This is the energy consumed multiplied by the number of clock cycles required to complete the program. There are similar metrics which emphasize the delay part of the equation to different degrees, but there is no consensus in literature about which is the best. Table 6 summarizes the execution time and energy consumed for each predictor strategy as a ratio to LEA+. The execution time and energy are averaged over the entire SPEC2000 integer suite.

Table 6: Energy and Execution Time Normalized to LEA+

	Execution Time	Energy
At-Fetch Stride	1.020	2.798
At-Fetch Hybrid	1.024	11.847
Post-Decode Stride Ld VP	1.026	0.223
Decoupled Gated	0.989	2.333
Decoupled Hybrid	0.988	5.924
LEA	1.007	0.562
LEA+	1.000	1.000

4.4 Improvement to Fetch-Time Prediction

Decoupled value prediction performs the best and LEA prediction is the most energy-efficient. The positive for traditional At-Fetch and Post-Decode predictors is a reduced set of data versus Decoupled value prediction. There are many fewer kilobytes being stored with an At-Fetch or Post-Decode value predictors since they do not require a Prediction Value Cache. If the energy consumption is reduced and the performance improved beyond the level of LEA predictors, these are still decent choices when area and high-performance are the design priorities.

Figure 11 examines the benefits of filtering instructions that access the read ports of an At-Fetch hybrid predictor. These results are based on the optimistic assumption that each instruction has access to its instruction type information as well as confidence counters for three types of port filtering - fast instruction filtering, mispredicted instruction filtering, and useless instruction filtering. This filtering data is assumed to be available as the instruction reads the value predictor (much like previous assumptions for At-Fetch and classifying predictors). The filters stop an instruction from accessing a read port. This allows more useful instructions to use the read ports. If no other instructions want the read ports, then energy is saved.

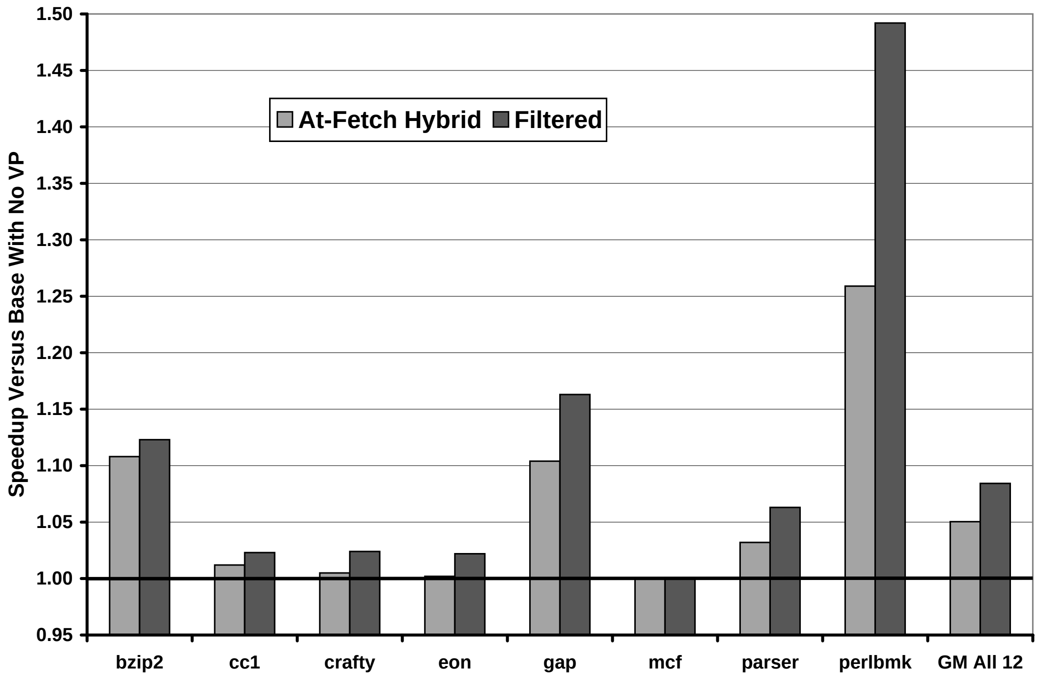


Figure 11: Filtering for At-Fetch Value Prediction

The first filtering method uses the instruction type information, allowing an instruction to access an At-Fetch predictor with the same information as the Post-Decode predictor (instructions beyond the first branch and exclusively result producing instructions). The fast instruction filter identifies quickly executing instructions. *Quickly executing* is defined as instructions that execute before a value prediction becomes available. The useless filter is based on the observation of "useless" predictions [22] and prevents instructions whose predictions are not consumed from doing more predictions.

The filters are driven by history-based counters. The counters for the fast filter and useless filter are the common two-bit saturating counters found in many types of speculative hardware. They are initialized to zero and filter with a counter value above or equal to two. The mispredict filter uses a three bit counter that only predicts when it is fully saturated at seven.

If this information is available at fetch time, combining the instruction type filter, fast filter, mispredict filter, and useless filter results in a significant improvement, 64.5% more speedup than the unfiltered At-Fetch Hybrid. The 8.4% speedup over the base is better than all of the previously studied strategies except for Decoupled and Unconstrained prediction. The filtering mechanisms also reduce read port activity, which in turn reduces dynamic predictor energy consumed. These three filters eliminate 42.4% of read accesses, dropping the overall dynamic value prediction energy by 18.3%.

5 Related Work

There have been many strategies proposed for value prediction. The primary ones include last value prediction [13, 14], stride prediction [8, 10], context prediction [24, 30], and hybrid prediction [22, 30]. More recently, hybrid predictions with the ability to dynamically classify instructions have been evaluated [23, 11]. In this work, we look primarily at a hybrid predictor with a last value predictor, stride predictor and a context predictor, similar to the predictors used in [23, 11] but without the dynamic classification schemes.

5.1 Reducing Size and Increasing Efficiency

There have been several proposals to reduce the size of value predictors, particularly hybrid predictors. it is possible to share hardware and use the given hardware more efficiently while maintaining or increasing performance. As shown in this work, hybrid predictors can be very expensive in size and power. So any efforts to reduce table size will certainly help both the energy and latency aspects of value prediction. However, the port problem still exists, and the reduced state often comes with an increase in access complexity and post-access logic, which add to predictor latency.

Burtscher and Zorn propose a hybrid load value predictor which performs better than predictors with more state [3], similar to the ones studied in this work. They are able to reduce the hybrid predictor size in half using state sharing and value compression techniques while increasing performance. Similarly, Pinuel et al. explored a more efficient hybrid value predictor which reduces state due to redundant predictions (e.g. last value predictor, stride predictor, and context predictor all unnecessarily do last value prediction) [18].

A very high performing context value predictor is the differential FCM predictor by Goeman et al. [2]. This predictor uses the FCM scheme for context prediction [24] to predict strides (difference in values) instead of the actual values. This strategy for prediction increases the efficiency of the tables, especially the second level table.

Tullsen et al. present a method of storageless value prediction [28]. Their method is based on exploiting locality of register values. By using values already in registers, there is no need to store these values in a value prediction table. Using static and dynamic approaches with compiler assistance, they show improvement over a 1k-entry last value predictor. While the energy requirements of this technique may be less than typical predictors, the performance also lags significantly versus high-performance value predictors.

5.2 Port Constrained Value Prediction

Calder et al. present techniques to determine when to use value predictions and which instructions to value predict [4]. They investigate limiting the instructions that update the predictor and the instructions that consume predicted values. This work is based on determining critical paths. Tune et al. continue this work with Critical Path Predictions which they apply to value prediction [29]. Their architecture allows only one value prediction per cycle (one read port). Critical Path Prediction is used to determine which instructions will use this resource. Fields et al. also propose a mechanism to isolate critical instructions, but they apply it to value prediction update [7].

Lee et al. propose a decoupled value prediction for processors with a trace cache [11]. Our Decoupled predictor is based on this model. They perform value predictions at instruction retire using a dynamic classification value prediction scheme based on a scheme by Rychlik et al. [23]. The goal of this work is to remove value prediction from the critical path in instruction fetch and to reduce the port requirements for the four value prediction tables. The architecture provide update queues and allow two ports for updating the predictor. Lee and Yew also propose a similar system for use with an instruction cache [12].

Gabbay and Mendelson study the effects of fetch bandwidth on value prediction [9]. The authors propose a highly-interleaved prediction table for trace cache processors to address the observations and issues uncovered in their work. Their simulated architecture uses idealized components to stress the instruction fetch and value prediction aspects of the work. They also measure the distance between producers and consumers in numbers of instructions. Similar to the issue to consume latency we measure in number of cycles.

5.3 Filtering Value Prediction Accesses

Calder et al. study confidence strategies and mechanisms to provide speedup using size-restricted value predictors [4]. Techniques studied include prediction confidence counters, replacement counters, and warm-up counters. In addition, they investigate limiting the instructions that update the predictor and the instructions that consume predicted values. The strategies presented include entirely eliminating non-loads and filtering updates from instructions with unconsumed predictions. They also look at identifying the longest path in the current instruction window and altering value prediction confidence thresholds based on the length of an instruction's current path.

Rychlik et al. analyze a hybrid predictor in a value prediction architecture with selective recovery. They also touch on the "usefulness" of value predictions. They observe that 31% of SPECint95 and 62% of SPECfp95 predicted values are never read before they are overwritten by the final result (a much higher percentage than we found for our base architecture and SPECint2000). They introduce a mechanism which only updates the value predictor on useful predictions. However, this mechanism did not improve prediction rates, while IPC increased very slightly.

5.4 Energy and Latency

There have been efforts to quantify the performance impact of latency-constrained tables for an entire processor [1] and fetch hardware [20], but there is no work of which we are aware that has applied these constraints to value predictors. Moreno et al. analyze power-related issues with value speculation without dealing with restricted ports or latency issues [16]. They present speedup versus power consumed for several configurations of value predictors. They note that a classifying hybrid predictor can potentially reduce the energy for value predictor reads. They also discuss the power issues due to mispredicted values and instruction window complexity.

6 Summary

In this paper, we take a close look at the viability of value prediction in a high-frequency, wide-issue microprocessor. Our initial analysis shows that 78-94% of load instructions and 73-99% of integer operations in the SPECint2000 benchmark suite have a potential consumer within one cycle. This indicates an urgency to break data dependencies. We also see that about one-third of integer operations execute in under three cycles. These are instructions that will not even be able to tolerate short unhidden value predictor latencies.

Based on this analysis we focused our studies on a) the performance impact on value predictors of table access latency and b) the energy consumed under these constraints when targeting high performance. We see that unhidden value predictor latency can cause noticeable performance degradation in traditional At-Fetch value predictor, up to a 62% reduction in speedup with a latency of 32 cycles. When carefully choosing the size of the table, number of ports, and associativity, the actual latency can be much less and the performance is tolerable.

At-Fetch, Post-Decode, Decoupled, and Latency and Energy Aware (LEA) predictors are explained and analyzed with restricted resources and table access latencies. In addition, a load-only value predictor is examined. Decoupled prediction provides 10.2% speedup over our baseline processor, the best performance among the studied predictors. Decoupled prediction benefits from low-latency high-bandwidth prediction access from a Prediction Value Cache. It incurs the value predictor table access latencies only at update time when the delay does not affect performance.

We propose LEA value prediction to address the complexity and energy consumption detriments of Decoupled prediction. The centralized, PC-indexed value prediction table is eliminated. LEA uses the original prediction traces and performs stride value prediction updates on the buffered prediction traces at instruction retire. These prediction traces are then stored directly to the Prediction Value Cache. LEA prediction reduces energy consumption by 58-95% versus the other complete value prediction strategies studied. Only LEA and the load-only value predictor consume less energy than the on-chip caches, with the load-only value predictor consuming the least (22% of LEA+ energy).

Recognizing the size advantage that traditional At-Fetch prediction has over Decoupled and LEA prediction, we introduce several possible history-based filters that result in 68% more speedup and 18% less dynamic energy consumption than a basic At-Fetch predictor. We are working on applying a LEA-like approach to allow the filter values to be delivered to an At-Fetch predictor in a timely and realistic manner.

Value prediction is important to increasing instruction level parallelism and improving the utilization of resources available in wide-issue microprocessors. In this work, we have shown that under technology constraints and energy analysis, value predictors have to be chosen with care to be viable, but that they can be a viable mechanism for achieving high performance in future wide-issue high-frequency processors.

References

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate versus IPC: The end of the road for conventional microarchitectures. In *27th International Symposium on Computer*

Architecture, pages 248–259, Jun 2000.

[2] H. V. B. Goeman and K. D. Bosschere. Differential FCM: Increasing value prediction accuracy by improving table usage efficiency. In *7th International Symposium on High Performance Computer Architecture*, Jan 2001.

[3] M. Burtscher and B. G. Zorn. Hybridizing and coalescing load value predictors. In *International Conference on Computer Design*, pages 81–92, Sep 2000.

[4] B. Calder, G. Reinman, and D. M. Tullsen. Selective value prediction. In *25th International Symposium on Computer Architecture*, pages 64–74, May 1999.

[5] R. F. Cmelik and D. Keppel. Shade: A fast instruction-set simulator for execution profiling. Technical Report SMLI 93-12 and UWCSE 93-06-06, Sun Microsystems Laboratories, Incorporated, and the University of Washington, 1993.

[6] R. J. Eickemeyer and S. Vassiliadis. A load-instruction unit for pipelined processors. *IBM Journal of Researh and Development*, 1993.

[7] B. Fields, S. Rubin, and R. Bodik. Focusing processor policies via critical-path prediction. In *28th International Symposium on Computer Architecture*, pages 74–85, Jul 2001.

[8] F. Gabbay and A. Mendelson. Speculative execution based on value prediction. Technical Report 1080, Technion - Israel Institute of Technology, Nov 1996.

[9] F. Gabbay and A. Mendelson. The effect of instruction fetch bandwidth on value prediction. In *25th International Symposium on Computer Architecture*, pages 272–281, June 1998.

[10] J. Gonzalez and A. Gonzalez. The potential of data value speculation to boost ILP. In *International Conference on Supercomputing*, pages 21–28, Jul 1998.

[11] S. Lee, Y. Wang, and P. Yew. Decoupled value prediction on trace processors. In *6th International Symposium on High Performance Computer Architecture*, pages 231–240, Jan 2000.

[12] S. Lee and P. Yew. On some implementation issues for value prediction on wide-issue ILP processors. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 145–156, Oct 2000.

[13] M. H. Lipasti and J. P. Shen. Exceeding the dataflow limit via value prediction. In *29th International Symposium on Microarchitectures*, pages 226–237, Dec 1996.

[14] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value locality and load value prediction. In *7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 138–147, Oct 1996.

[15] S. McFarling. Combining branch predictors. Technical Report TN-36, Digital Western Research Labs, Palo Alto, Calif., Jun 1993.

[16] R. Moreno, L. Pinuel, S. del Pino, and F. Tirado. A power perspective of value speculation for superscalar microprocessors. In *International Conference on Computer Design*, pages 147–154, Sep 2000.

[17] S. J. Patel. *Trace Cache Design for Wide-Issue Superscalar Processors*. PhD thesis, The University of Michigan, 1999.

[18] L. Pinuel, R. Moreno, and F. Tirado. Implementation of hybrid context-based value predictors using value sequence classification. In *4th Euro-Par Conference*, Aug-Sep 1999.

[19] A. Ramirez, J. Larriba-Pey, C. Navarro, J. Torrellas, and M. Valero. Software trace cache. In *International Conference on Supercomputing*, pages 119–126, Jun 1999.

[20] G. Reinman, T. Austin, and B. Calder. A scalable front-end architecture for fast instruction delivery. In *26th International Symposium on Computer Architecture*, pages 234–245, May 1999.

[21] G. Reinman and N. Jouppi. An integrated cache timing and power model, 1999. COMPAQ Western Research Lab.

[22] B. Rychlik, J. Faistl, B. Krug, and J. P.Shen. Efficacy and performance impact of value prediction. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 148–154, Oct 1998.

[23] B. Rychlik, J. W. Faistl, B. P. Krug, A. Y. Kurland, J. J. Sung, M. N. Velev, and J. P. Shen. Efficient and accurate value prediction using dynamic classification. Technical report, Carnegie Mellon University, 1998.

[24] Y. Sazeides and J. E. Smith. The predictability of data values. In *30th International Symposium on Microarchitecture*, pages 248–258, Dec 1997.

[25] Semiconductor Industry Association. The national technology roadmap for semiconductors, 1999.

[26] M. Smotherman and M. Franklin. Improving CISC instruction decoding performance using a fill unit. In *28th International Symposium on Microarchitecture*, pages 219–229, Nov 1995.

[27] Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmarks. http://www.spec.org/osg/cpu2000/.

[28] D. M. Tullsen and J. S. Seng. Storageless value prediction using prior register values. In *25th International Symposium on Computer Architecture*, pages 270–279, May 1999.

[29] E. Tune, D. Liang, D. M. Tullsen, and B. Calder. Dynamic prediction of critical path instructions. In *7th International Symposium on High Performance Computer Architecture*, Jan 2001.

[30] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In *30th International Symposium on Microarchitecture*, pages 281–290, Dec 1997.

[31] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.