Latency and Energy Aware Value Prediction for High-Frequency Processors

Ravi Bhargava and Lizy K. John
The University of Texas at Austin
Department of Electrical and Computer Engineering
Austin, Texas
{ravib,ljohn}@ece.utexas.edu

ABSTRACT

This work addresses the issues of access latency and energy consumption in value predictor design for high-frequency, wide-issue microprocessors. Previous value prediction research allows for generous assumptions regarding table configurations and access conditions, while ignoring prediction latencies and energy issues. However, the latency of a high-performance value predictor cannot always be completely hidden by the early stages of the instruction pipeline as previously assumed, and it causes noticeable performance degradation versus unconstrained value prediction. This paper describes and compares several variations of basic value prediction methods: at-fetch, post-decode, and decoupled.

The performance of at-fetch and post-decode value predictors is limited by the high access latency of accurate predictor configurations. Decoupled value prediction excels at overcoming the high-frequency table access constraints by placing completion-time predictions into a separate and easily accessible storage. However, it has high energy requirements. We study 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 a queue of prediction traces. This latency and energy aware method of maintaining and distributing speculated values leads to a 58%-95% reduction in value predictor energy consumption versus known value prediction techniques while still maintaining high performance.

Categories and Subject Descriptors: C.1.1 [Processor Architectures]: Single Data Stream Architectures

General Terms: Design, Measurement, Performance

Keywords: data speculation, low power, complexity-effective design, trace cache processors

ICS'02, June 22-26, 2002, New York, New York, USA.

1. INTRODUCTION

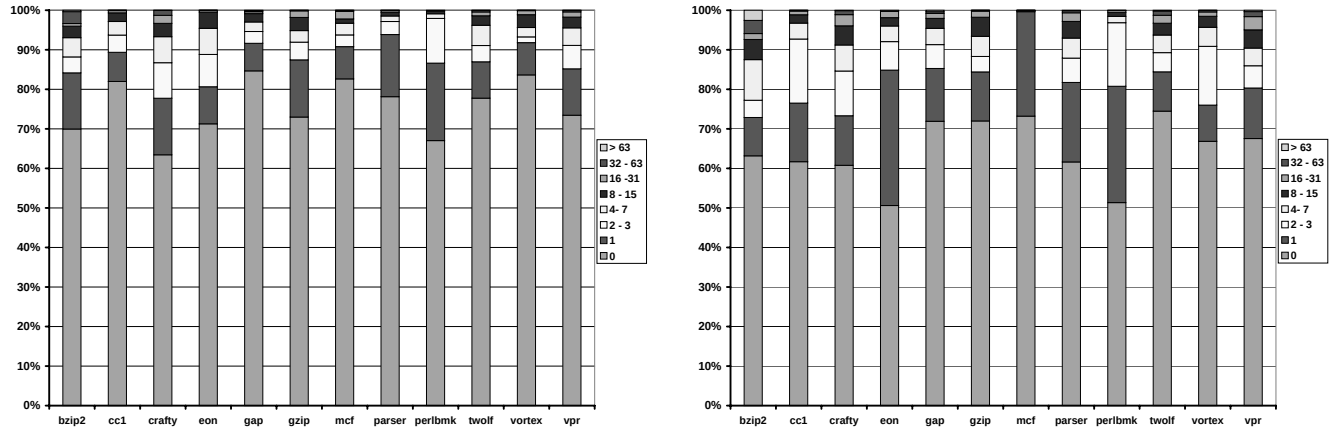
One trend in recent microprocessor design is the increasing density of on-die transistors. This has allowed designers and researchers to explore creative new techniques for improving instruction throughput. One strategy for utilizing this wealth of transistors is to increase execution resources and speculation hardware. Value prediction is one technique that has shown great potential for improving instruction throughput. True data dependencies are a fundamental obstacle to higher instruction level parallelism (ILP), and value prediction allows these dependencies to be satisfied early by speculating on instruction results. Instructions are then free to execute in parallel, leading to higher utilization of the processor resources. This is especially useful in wide-issue microprocessors where execution resources are often underutilized.

Urgency To Predict Values: 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, the producer-consumer distance is measured in clock cycles for a specific implementation of a wide-issue microarchitecture. The cycles are measured from instruction issue to the first *request* from an issued consumer instruction.

These graphs depict 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 more quickly as the achievable fetch and issue widths increase. If values cannot be predicted swiftly enough to break these data dependencies, value prediction will be less useful.

Table Access Latency: The push toward high frequencies in microprocessors creates a processor constrained by wire delay, 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 most effective.

The graph in Figure 2 illustrates the access times for value predictors of several sizes versus the number of ports. The smallest structure, a 1024-entry predictor with one port, has a two cycle access latency. As the table size increases, the impact of additional ports becomes more severe. A similar analysis shows that port related delays dominate to the point that increasing associativity causes little additional



a. Loads b. Integer Operations

Figure 1: Distribution of Issue to Consume-Request Distances Measured in Clock Cycles
The benchmarks are the SPEC CPU2000 integer benchmarks presented in Section 3. The processor configuration for this graph is our baseline microarchitecture described in Section 3.1.

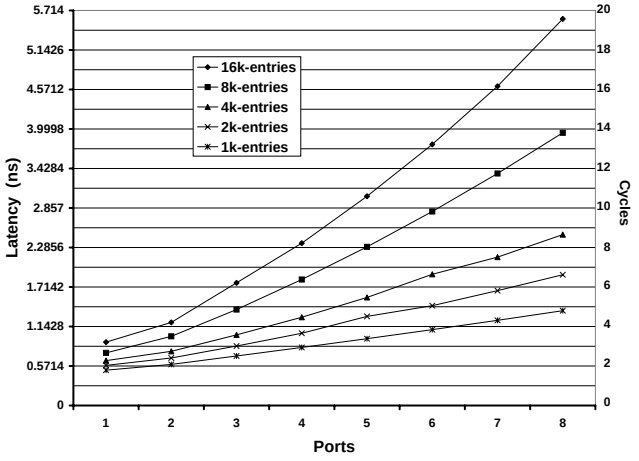


Figure 2: Access Latency Versus Number of Ports
A next generation processor [25] with a 100nm technology and a 3.5 GHz clock frequency is modeled. Each horizontal line on the graph represents one clock period (0.2857ns) at 3.5 GHz. These latencies are obtained using Cacti 2.0 [21]. We use the minimum 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.

delay [3].

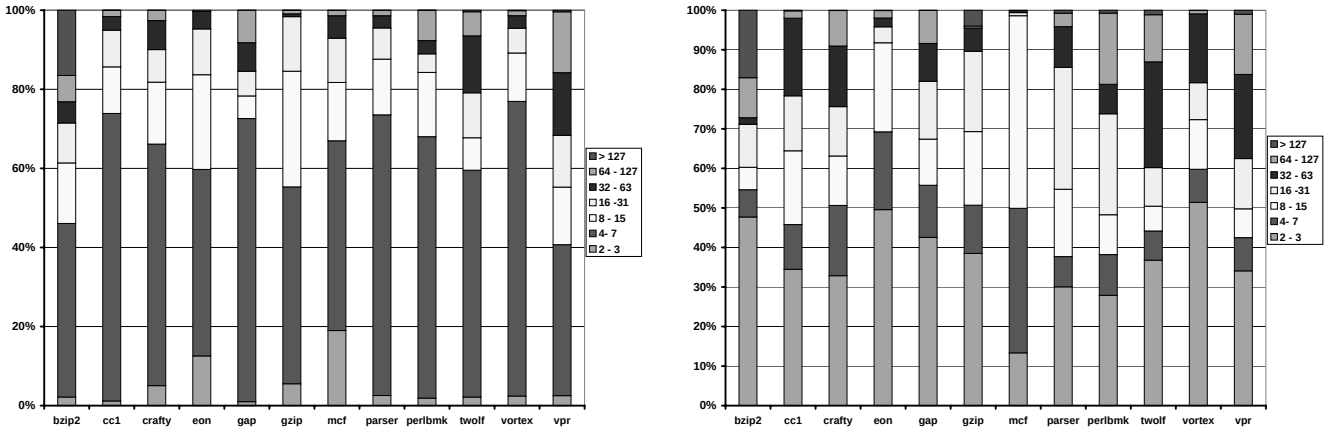
Value predictors in literature do not typically account for the table access latency. However, this latency can quickly become an issue in reasonable predictor configurations. For example, consider a two-level context value predictor for a wide-issue microprocessor where each table is direct-mapped with 8192 entries. Based on the latencies in Figure 2, accessing such a context predictor (8-ports for each table) takes 28 total cycles, 14 cycles for each table. These values do not even 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 illustrate the distribution of instruction execution latencies for load instructions 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 perceived execution latency can be reduced or even eliminated by timely and successful value prediction. However, there is no benefit to speculating on an instruction value if the prediction latency exceeds the execution latency.

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 benefits 81-99% 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.

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. 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 from and write to this structure to maintain high performance in a wide-issue microprocessor. In the SPEC CPU2000 integer benchmarks



a. Loads b. Integer Operations

Figure 3: Distribution of Rename to Result Times Measured in Clock Cycles
The benchmarks are the SPEC CPU2000 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.

analyzed in this work, 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. The number of ports is varied for different numbers of table entries in Figure 4.a, while associativity is varied in Figure 4.b. Increasing either design factor is expensive in terms of energy, but note that the energy scale for associativity (number of ports fixed at four) is more than double that of the ports graph. So while access latency is dominated by delay due to ports, energy consumption is more sensitive to the associativity. These graphs solidify the notion that straightforward steps to increase performance can cost designers in energy consumption.

A value predictor may not dominate overall energy consumption, but our results confirm that it can contribute non-trivially. Since a value predictor is read and updated by multiple instructions each cycle, 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 at-fetch hybrid predictor (as described in Table 3) consumes almost 10 times more energy than all of the on-chip caches combined! Although on-chip caches store large amounts of data, the energy consumed is not dramatic since the number of ports is typically limited to one or two. In addition, the cache port accesses are 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. In this work, we study how value prediction will perform in a wide-issue, high-frequency environment with realistic constraints on the number of read and write ports, table access latencies, and energy requirements. 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 form of value prediction that reduces value prediction complexity and energy consumption while maintaining high performance. In addition to the analysis in this section, the contributions of this work include:

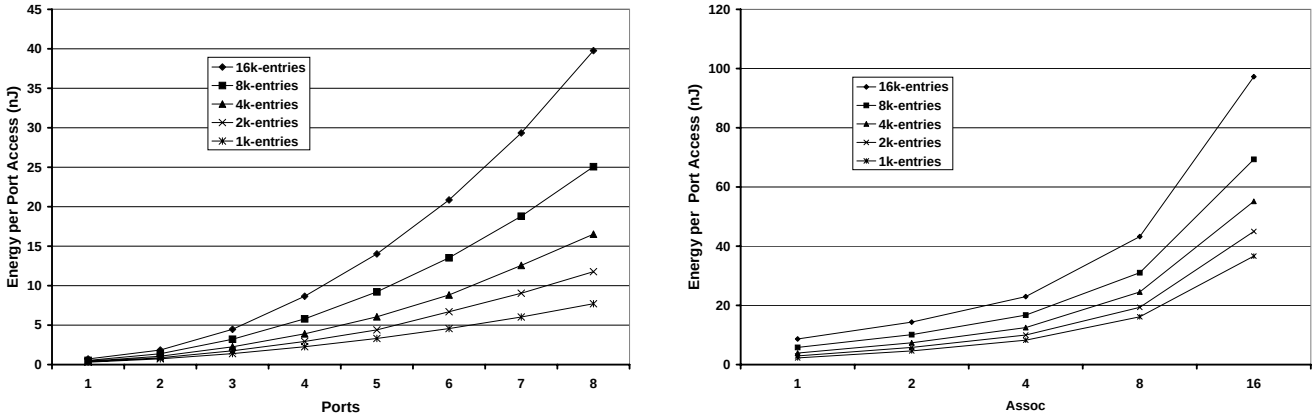
1. Studying the sensitivity of value prediction speedup to access latencies and port considerations. The performance gained from increasing the number of ports is tempered by the resulting increase in latency.

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 trace-based reads from a specialized cache (as in decoupled prediction) and trace-based updates of buffered prediction traces.

3. Comparing performance of several methods and configurations of value prediction in the presence of latency considerations. We compare the LEA predictor to hybrid and stride versions of the traditional at-fetch value predictor and post-decode value predictor, as well as a decoupled value predictor. The decoupled predictor provides the best performance, achieving 10.2% speedup versus the base architecture with no value prediction.

4. Extending the analysis to account for the energy for a variety of value predictors. LEA value prediction reduces the dynamic table access energy by 58%-95% while achieving 8.6% speedup versus the base architecture.

The remainder of this paper is organized as follows. Section 2 explains at-fetch, post-decode, decoupled, and LEA value predictions. Section 3 describes our evaluation methodology. Section 4 compares several value predictor strategies with respect to performance and energy. Section 5 discusses related value predictor work. Section 6 summarizes and concludes the paper.



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.

2. VALUE PREDICTOR PLACEMENT AND IMPLEMENTATION CHOICES

We examine multiple points for predicting an instruction value. Two basic methods are seen in the 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 prediction as a back-end option.

2.1 At-fetch Value Prediction

Performing value prediction at instruction fetch is a commonly assumed implementation [5, 10, 14, 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 beyond 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 for acquiring multiple predictions with a single access that may work when fetching up to the first branch, such as banking, no longer work well [10]. The second restriction is the lack of instruction type information. During fetch, the instruction types are indistinguishable, so value prediction resources are being consumed by instructions that are not eligible for predictions (branches, stores, floating point operations). This proves costly for a port-constrained predictor.

The advantage of at-fetch prediction is compelling. In general, there is no need for a predicted value until the instruction has been decoded and renamed. Therefore, in at-fetch prediction 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 any non-contiguous instructions beyond the first branch are 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 type of post-decode information is commonly (and optimistically) assumed for at-fetch predictors.

The disadvantage of post-decode prediction is that the value predictor access latency can no longer be hidden by earlier stages of the pipeline. Value predictions are being initiated at the point 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 enter the instruction window.

2.3 Decoupled Value Prediction

Predicting at fetch time or after instruction 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 [12] and one for machines with no trace cache [13]. The trace cache strategy is used in this paper since it fits perfectly within the wide-issue environment being evaluated. 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. This is similar to cache banking in that one fetch provides multiple predictions. 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 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 update produces 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 (e.g. 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 a queue of 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 than value prediction tables because 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 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 traditional hybrid prediction, are no longer feasible without requiring extra hardware and 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, we have seen that trace-like storage is often an advantage due to correlation and context effects.

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

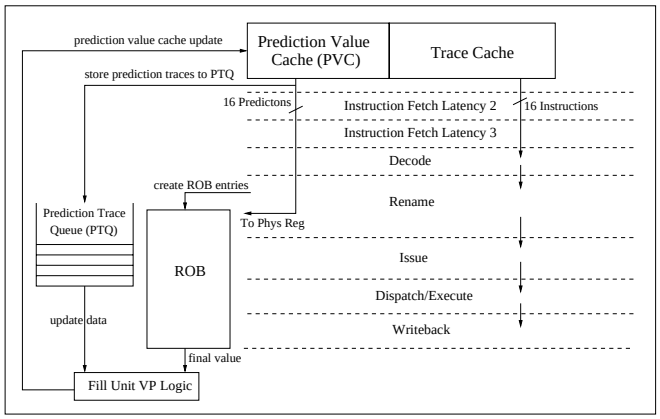


Figure 5: Overview of Latency and Energy Aware Value Prediction

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.

3. METHODOLOGY

In this section, we describe the benchmarks and simulation environment used to study value predictor trade-offs. We examine the integer benchmarks from the SPEC CPU2000 suite [27]. The SPEC programs are dynamically linked and compiled using the Sun C 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.

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 [6]. 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

Table 1: SPEC CINT2000 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

frequency in all simulations (3.5 GHz in 100nm technology), 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-cyc hit, 30-cyc 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
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, 31]. The trace cache access latency is noticeably lower because it has just one read/write port. The configurations of the remaining cache structures are intentionally aggressive to place a little extra pressure on the instruction window and execution resources. 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 less prominent.

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. This is primarily presented for comparison purposes. All value predictors are assumed to be fully pipelined and capable of serving new requests every cycle. Since the clock frequency is fixed, the access latency is altered if necessary when the value predictor configurations are changed. The modeled latency from Cacti is divided by the clock period (0.2857 ns) to produce the latency. This is an optimistic assumption, partly because pipelining overheads are not considered.

For all value predictors, each table is direct-mapped, tagged[2], and updated by every result-producing instruction. When a value misspeculation is encountered, the microarchitecture reissues all instructions younger than the mispredicted instruction.

The number of table entries for the context predictor is chosen 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 4096-entry stride predictor, there is no reasonable two-level configuration that can be accessed in parallel.

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 reading 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. Update latencies are modeled (16 cycles for all predictors), but we find that update latencies have negligible effects on overall performance.

The two-level value predictor used in hybrid prediction is similar to the one discussed by Wang and Franklin [30]. The stride predictors studied use matched (two-delta) stride prediction [7]. The stride predictor uses a two-bit saturating counter for its internal prediction confidence mechanism. Each sub-predictor in the hybrid predictor 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 the indicated hybrid predictor simulations, we simulate a perfect confidence mechanism. In this case, if any of the sub-predictors produce the correct value, then this value is chosen. Even with a perfect confidence mechanism, the value predictor can speculate incorrectly if none of the sub-predictors produce the proper value. More details on the value predictor configurations can be found in our accompanying report [3].

[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 more accurately represent our simulated predictors.

Table 3: Table Access Parameters for 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	-	4096	-	4	5 cycle	1 cycle
Post-decode Hybrid	8192	8192	1024	4	8 cycles	8 cycles
Post-decode Stride	-	4096	-	4	5 cycles	5 cycles
Decoupled	8192	8192	8192	4	3 cycles	0 cycles
LEA/LEA+	-	-	-	-	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.

4. ANALYSIS

This sections first presents an analysis of value predictor sensitivity to latency, access ports, and predictor size. Understanding these effects, a comparison of the different value prediction strategies is performed. After analyzing the predictors strictly from a performance point of view, we study the energy implications of each configuration.

For aesthetic reasons, in the speedup figures, the geometric mean of all 12 SPEC CINT2000 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 geometric mean.

4.1 Effects of Table Access Latency

This section analyzes the impact of value predictor latency on performance. Figure 6 presents speedup as the access latency is altered for a version of the `Unconstrained` predictor with 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. The assumption in previous value predictor work is that unhidden latency is zero.

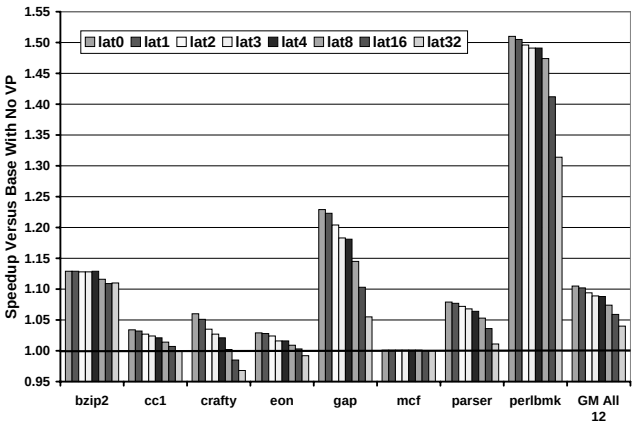


Figure 6: Effect of Unhidden Table Access Latency
`lat` is the number of cycles after an instruction is decode that its predicted value becomes available.

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 to predictor access 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 provides noticeable speedup.

Factors that have a large effect on 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 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 entry.

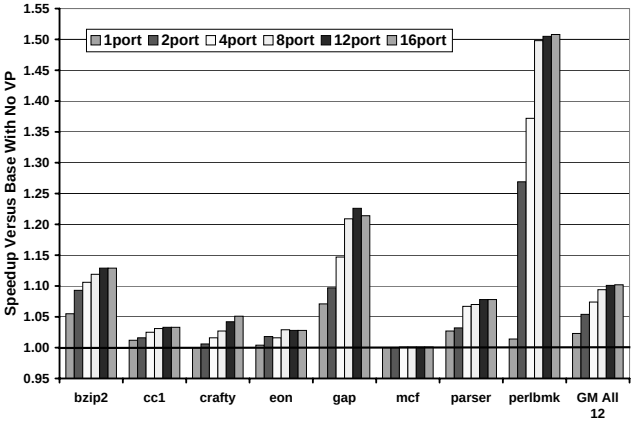


Figure 7: Effect on Performance of Restricted Ports
Ports are expressed in the number of read and write ports. This is modeled in Cacti as two read/write ports and read and write port pairs after two.

Figure 7 illustrates that reducing read ports to 12 in a 16-wide environment affects performance very little. 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.

In Figure 8, the number of table entries is varied 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 8192-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, 4096-entry and 8192-entry tables are attractive.

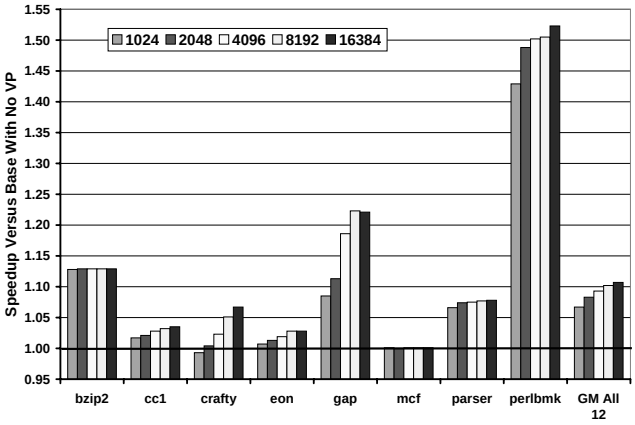


Figure 8: Effect on Performance of Number of Table Entries

4.2 Comparing Prediction Strategies

In this section, performance is presented for specific implementations of the value prediction strategies covered in this work. The analysis in Section 4.1 shows that tables with eight ports incur high access times and energy consumption, while those with two ports lead to a significant performance decrease. Tables with less than 4096 entries are too small for high performance, but those with more than 8192 entries provide little benefit given the energy and latency penalty. Therefore, we focus on two port/size configurations, 4096-entry tables with four ports and 8192-entry tables with four ports (as discussed in Section 3.2, Table 3).

Figure 9 compares the overall performance of each value prediction strategy and latency-constrained configuration. In this section, the `Unconstrained`, at-fetch, post-decode, and decoupled hybrid predictors use perfect confidence for choosing among the sub-predictors. Two versions of LEA prediction are also analyzed. One performs matched stride prediction (`LEA+`) while the other performs basic stride prediction (`LEA`).

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 `Unconstrained`, but does achieve 77% of its speedup. 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 at-fetch value predictors are often modeled with assumptions similar to that of the unconstrained predictor.

The post-decode hybrid predictor is able to match the performance of the at-fetch hybrid predictor. This is somewhat surprising since post-decode predictors cannot hide access latency like at-fetch prediction. However, post-decode predictors perform more successful predictions as a result of better read port utilization. Even for small predictors whose latency can be fully hidden in the at-fetch scenario, post-decode predictors perform comparably or even better.

The performance superiority of the decoupled and LEA value predictors is not surprising since there is intrinsically no unhidden latency. In addition, the prediction read bandwidth is effectively unconstrained by ports because the predictions are in a trace format. These performance results highlight the advantage of performing completion time predictions and acquiring multiple predictions with one access.

The advantage of decoupled prediction over LEA prediction is the result of performing hybrid prediction versus 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 has high energy consumption) is required for the at-fetch and post-decode strategies to realize a performance advantage over predictors that generate prediction information at instruction retire.

4.3 Energy Analysis

In addition to performance enhancement, it is important to understand the energy requirements of value predictors. Figure 10 reports relevant port access occurrences for a predictor similar to `Unconstrained`, but restricted to four ports. Each port access consumes energy, so one important aspect of low-energy value prediction is to reduce these counts.

The first column for each benchmark is the number of port accesses that take place during execution due to value predictor reads (`VP Read`). Only load and integer instructions access the predictor since it is a post-decode predictor. The next column, `VP Update`, is the number of port accesses made at instruction retire due to updates to the value predictor. The `TR Read` column is the number of traces read. 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 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 predictable instructions that do not obtain access to a read port. All eligible instructions perform updates whether or not they read a prediction. `VP Read` is greater than `VP Update` when a large number of wrong-path instructions access the read ports. These instructions do not retire and therefore do not perform updates.

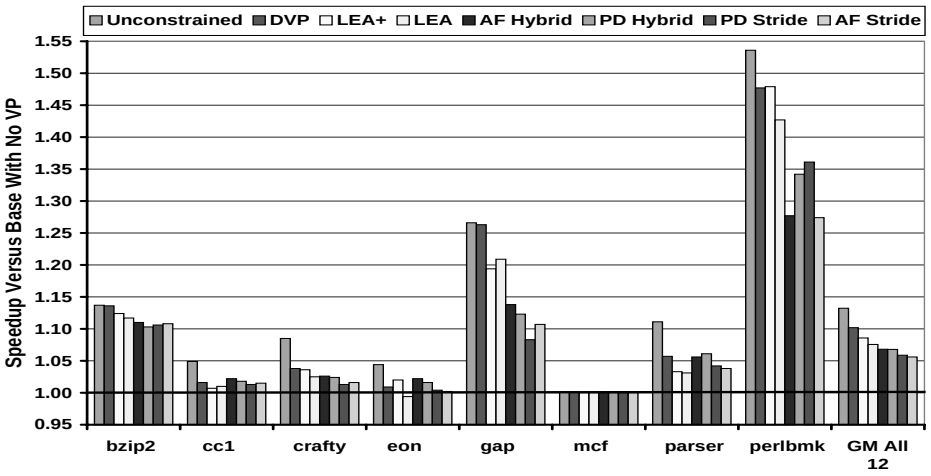


Figure 9: Comparison of Prediction Strategies
`DVP` is decoupled value prediction. `AF` is at-fetch value prediction. `PD` is post-decode value prediction. The at-fetch, post-decode, and decoupled hybrid predictors use perfect confidence for choosing.

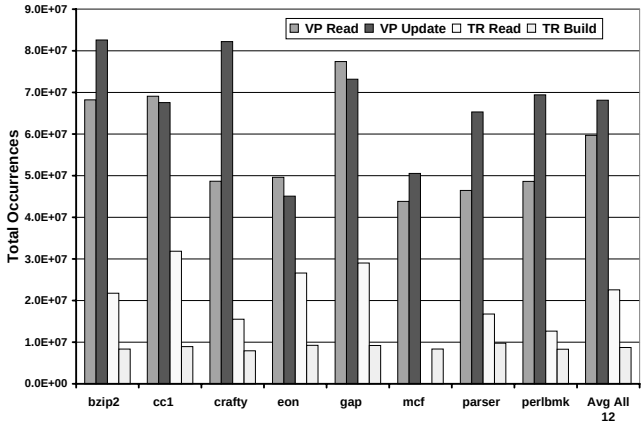


Figure 10: Table Port Accesses
The predictor used is a four-ported hybrid post-decode predictor with 8192-entries per table and a 0 total access latency.

Table 4 outlines all of the value prediction hardware used in the evaluated predictors. For instance, the at-fetch hybrid predictor discussed in the previous sections consists of the two levels of the context sub-predictor each with a 1024-entry table, the 8192-entry table of the stride sub-predictor, and the 8192-entry table of the LVP sub-predictor. On the other hand, the LEA predictor requires the 1024-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. To determine the total energy for each prediction strategy, we use port access counts (similar to Figure 10) for each predictor and then multiply those totals by the corresponding energy per port access from Table 4. It is assumed that the energy to perform a write is equivalent to the energy to perform a read.

The equations used for determining the overall energy of our value prediction strategies are presented with Table 4. The term in the `Energy Notation` field is the variable used in the equations to represent port access energy for the particular hardware structure. The subscript E stands for energy. The variable $hybrid_E$ is the energy required to access a hybrid predictor, defined directly below the Equation 6. The terms `VP Read`, `VP Update`, `TR Read`, and `TR Build` have the same meaning as in the previous discussion of Figure 10.

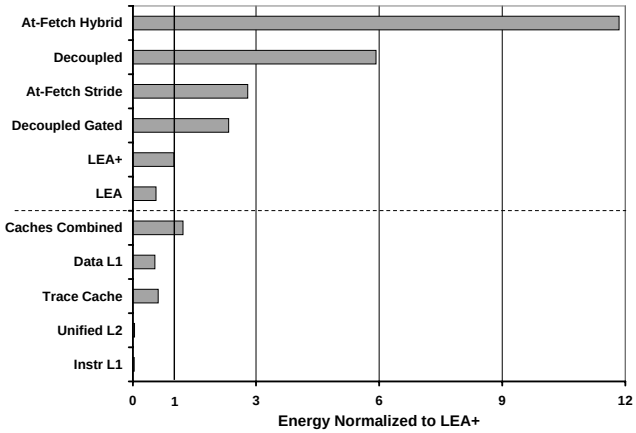


Figure 11: Energy Consumption of Value Predictors and On-chip Caches.
`Caches Combined` is the Data L1, Trace Cache, Instruction L1, and Unified L2 combined energy

To put the energy consumption of value prediction into perspective, Figure 11 compares the dynamic access energy of value prediction to the energy of the L1 data cache, L1 instruction cache, L2 cache, and trace cache from our baseline microarchitecture. Moreno et al. [16] present a classifying hybrid predictor where only one prediction table is "turned on" during a value prediction. Taking this possibility into account, `Decoupled Gated` is also presented. In this configuration, only the stride predictor energy is consumed and the other tables are assumed to consume no energy.

Studying just the value predictor energy results, the hybrid predictors (`At-Fetch Hybrid` and `Decoupled`) consume more energy than other configurations. The stride predictor versions of at-fetch and the gated decoupled predictors

Table 4: Energy Parameters for Value Predictor Structures

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

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) $LEA{+}_E$ = (TR Read * $LEA{+}PVC_E$) + (TR Build * $LEA{+}PVC_E$) + (2 * TR Read * $LEA{+}PTQ_E$)
(6) LEA_E = (TR Read * $LEAPVC_E$) + (TR Build * $LEAPVC_E$) + (2 * TR Read * $LEAPTQ_E$)
...where $hybrid_E$ = ($8kTable_E$ + $8kTable_E$ + $1kCtxtL1_E$ + $1kCtxtL2_E$)

do significantly better, consuming 4.1 and 2.8 times less energy. However, LEA prediction is a far more energy-efficient method for 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 value predictor. LEA with basic stride prediction has smaller entries and is even more energy efficient, reducing energy by almost one-half versus LEA+. This energy analysis highlights the primary benefits of LEA prediction.

The lower portion of Figure 11 presents the energy consumed by the on-chip caches. An at-fetch hybrid predictor consumes 14 times as much energy than 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 on-chip caches. The reduction in energy is a direct result of the value predictor ports being accessed more often 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 a fetch occurs from either the instruction cache or the trace cache data, but never both on the same fetch. Finally, only the LEA predictors has similar energy consumption as the caches.

In an extended version of this work [3], load-only value prediction is found to have lower energy requirements than LEA prediction. A two-ported, post-decode, 4k-entry stride load value predictor is analyzed, and it consumes 22% of the energy of LEA+ value prediction. However, it only produces 4.5% speedup, lowest of the predictors analyzed. Even under optimal conditions (similar to `Unconstrained`), load value prediction only allows for 6.2% speedup for these benchmarks.

5. RELATED WORK

There have been many strategies proposed for data value prediction. The primary ones include last value prediction [14, 15], stride prediction [9, 11], context prediction [24, 30], and hybrid prediction [22, 30]. More recently, hybrid predictors with the ability to dynamically classify instructions have been evaluated [12, 23]. In this work, we look primarily at advanced hybrid predictors with a last value predictor, stride predictor and a context predictor, 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 improve table efficiency 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 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 [4] propose a hybrid load-only value predictor which performs better than predictors with more state, 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. [18] explored a more efficient hybrid value predictor that exploits redundant predictions (e.g. last value predictor, stride predictor, and context predictor all unnecessarily do last value prediction).

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 1024-entry last value predictor. While the energy requirements of this technique may be less than typical predictors, the performance lags that of high-performance value predictors.

5.2 Port Constrained Value Prediction

Calder et al. [5] present techniques to determine when to use value predictions and which instructions to value predict. 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 [8].

Lee et al. propose a decoupled value prediction for processors with a trace cache [12]. 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.

Gabbay and Mendelson study the effects of fetch bandwidth on value prediction [10]. 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 which we measure in number of cycles.

5.3 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 SPEC CINT2000 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. Even value predictors with short unhidden access latencies will not benefit these instructions.

Based on this analysis, the study focuses on a) the performance impact of table access latency on value predictors 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 a 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) value predictors are explained and analyzed with restricted resources and table access latencies. Decoupled prediction provides 10.2% speedup over our baseline processor, the best performance among the studied predictors. It benefits from low-latency high-bandwidth prediction access from a Prediction Value Cache, and 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 shortcomings of decoupled value prediction. Centralized, PC-indexed value prediction tables are eliminated. LEA uses the original prediction traces and performs stride value prediction updates on the buffered prediction traces at instruction retire in the fill unit. These prediction traces are then stored directly to the Prediction Value Cache. LEA prediction reduces energy consumption by 58-95% versus the other value prediction strategies studied and is the only method that consumes less energy than the on-chip caches.

Value prediction is an important technique for increasing instruction level parallelism and improving the utilization of the resources available in wide-issue microprocessors. In this work, we have shown that under technology and energy constraints, value predictors must be chosen with care to be useful. With the proper latency and energy considerations, value prediction can be a productive mechanism for achieving high performance in future wide-issue high-frequency processors.

7. ACKNOWLEDGMENT

The authors thank the anonymous reviewers as well as Juan Rubio and Madhavi Valluri of the LCA for their insightful comments and suggestions. Ravi Bhargava is currently supported by an Intel Foundation Graduate Fellowship Award. This research is also supported in part by the State of Texas Advanced Technology program grant, by the National Science Foundation under grant numbers ECS-0113105 and EIA-9807112, and by Microsoft, Tivoli, Motorola, Intel, and IBM.

8. 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] R. Bhargava and L. K. John. Value predictor design for high-frequency microprocessors. Technical Report TR-020508-01, The University of Texas at Austin, Laboratory for Computer Architecture, May 2002. http://www.ece.utexas.edu/projects/ece/lca.

[4] M. Burtscher and B. G. Zorn. Hybridizing and coalescing load value predictors. In *International*

Conference on Computer Design, pages 81–92, Sep 2000.

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

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[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.