Evaluating Trace Cache Energy-Efficiency

University of Virginia Dept. of Computer Science Technical Report CS-2004-31

Michele Co and Kevin Skadron
Department of Computer Science
University of Virginia
{micheleco, skadron}@cs.virginia.edu

Abstract

Future fetch engines need to be energy-efficient. Therefore, a thorough evaluation and comparison of fetch engine design is necessary for futuristic processors.

Our work compares the energy-efficiency of concurrent trace caches (CTCs), sequential trace caches (STCs), block-based trace caches (BBTCs), and instruction caches (ICs). We compare: CTCs and STCs with path-based next trace predictor (NTP), ICs with branch predictor (*IC-BPRED*), and BBTCs with trace table (*BBTC-TT*). To separate out predictor organization and prediction effects we also evaluate ICs with NTP (*IC-NTP*) and BBTCs with NTP (*BBTC-NTP*). In our experiments, we first evaluate the fetch engines with no area budget restrictions. Then, to consider higher clock rates we evaluate the fetch engines when restricting the area budget for each component. To consider future process technologies, we also evaluate the effect of increased leakage.

We find that branch prediction (whether explicit or implicit) is a key component in the energy-efficiency of the fetch engine designs evaluated. Branch prediction effects are eliminated by artificially equalizing the effective branch prediction accuracy for the fetch engine designs and the results are evaluated.

We find that access delay limits the theoretical performance of the fetch engines evaluated. We propose a novel ahead pipelined NTP that performs nearly as well as the single-cycle access NTP.

1 Introduction

Energy-efficiency has become important for almost all new chip designs. For high-end processors, power density is a problem. Today's desktop CPUs become tomorrow's laptop CPUs so evaluating the energy-efficiency of microarchitectural designs is important. Furthermore, energy-efficiency is also important for wall-powered systems such as server racks in data centers where electricity and air conditioning are major costs. The fetch unit contributes a large portion of total power consumption in a microprocessor. For example, Montanaro *et al.* [16] measure the StrongARM's fetch engine power consumption at 27% of total chip power. Trends in branch prediction research also point toward larger and more aggressive fetch engine organizations [11, 20]. Understanding how fetch organization affects processor energy-efficiency is important to processor design.

The fetch unit's role is to feed the dynamic instruction stream to the execution unit. Instruction caches store instructions in static program order. Due to the presence of taken control flow instructions, some of the instructions fetched from the instruction cache are unused.

Trace caches store instructions in dynamic program order. Most trace cache implementations [14, 19, 22, 24] do not suffer from the problems of requiring additional levels of indirection or the need for interleaving or complex alignment networks and thus are options to be considered in fetch engine design. We are not aware of any work analyzing the energy-efficiency of trace caches compared to conventional fetch organizations. Our work models several types of trace caches: the conventional or concurrent trace cache (CTC) in which trace cache and instruction cache are probed in parallel, the sequential trace cache (STC) described by Rotenberg *et al.* [22, 24] and the block-based trace cache (BBTC) described by Black *et al.* [2]. We present four sets of experimental results.

First, we compare the following fetch engine organizations with and without area restrictions: CTC with NTP, STC with NTP, IC with hybrid branch predictor, BBTC with trace table trace prediction, and to separate trace prediction effects, IC with NTP and BBTC with NTP. In the first experiment, the fetch engine components have no restricted area budget. In the second experiment, to account for the trend of decreasing access times, each component in each fetch engine organization is limited to a restricted area budget. Second, to eliminate branch prediction effects we artificially equalize the branch prediction accuracy for all the fetch engine designs. Third, we examine the effect of increasing leakage on the fetch engine organizations. Finally, we introduce and evaluate an ahead pipelined NTP to address decreasing cycle times.

Each comparison is made with respect to two parameters: performance (IPC) and energy-delay-squared (ED^2).

The rest of the paper is organized as follows: Section 2, presents related work, Sections 3 and 4 present experimental methodology, Sections 5 through 6 present experimental results and Section 8 presents conclusions and directions for future work.

2 Related Work

Friendly, Patel, and Patt [7] and Rotenberg, Bennett, and Smith [22, 24, 23] performed comprehensive studies of the trace cache design space with respect to performance. We perform a similar design space study to evaluate power, energy, and performance tradeoffs on a more current processor pipeline.

Research has explored ways to reduce the power dissipation of trace caches. Hu *et al.* [9] showed that sequentially accessing the trace cache and instruction cache has significant power savings over accessing the two structures simultaneously. In subsequent work, Hu *et al.* [8] also compared the conventional trace cache (CTC), sequential trace cache (STC), and a new design, the dynamic direction prediction based trace cache (DPTC) for power efficiency and performance. They found that DPTC exhibits less performance loss than the STC but with similar power consumption. Our work compares fetch units containing either STCs or BBTCs to fetch units containing only an instruction cache and evaluates the effect of additional parameters such as leakage and delay.

Bahar [1], Kim [15], and Zhang [35] have done work to improve traditional instruction cache energy consumption without adversely affecting processor performance or on-chip energy consumption. Our work focuses on evaluating high fetch bandwidth fetch organizations as opposed to techniques to improve traditional instruction cache energy-efficiency.

Solomon *et al.* [30] introduced the micro-operation cache (UC) as an alternative frontend for the Intel P6 processor family. The UC stores basic blocks in decoded μop form and provides similar fetch bandwidth at lower power consumption. Their goal was not to increase fetch bandwidth but rather to find a more energy-efficient fetch engine design with comparable performance. Our work focuses on evaluating the energy-efficiency of fetch engine designs which seek to increase fetch bandwidth. Therefore, the UC is not included in this work.

Parikh *et al.* [18] explored the role of branch predictor organization on power, energy, and performance tradeoffs for fetch engine design. They found that although extra power may need to be expended for the branch predictor, overall processor power and energy dissipation can be reduced. Our work focuses on the cache and prediction mechanisms in various fetch engine organizations.

Ramirez *et al.* [21] introduce instruction *stream fetch engine* as a high-performance fetch mechanism. An instruction stream is a sequence of instructions which may contain only not-taken branches. This fetch unit takes advantage of code layout optimizations. We believe that instruction streams are a special definition of trace that falls somewhere in the spectrum between instruction cache lines (program ordered instruction blocks) and instruction traces (dynamic instruction sequences) and that some useful insights can be drawn without evaluating fetching instruction streams. Since we do not consider code layout optimizations for the fetch engines we evaluate, we do not evaluate fetching instruction streams because the limitation might unfairly penalize that approach. Therefore, the *stream fetch engine* is outside the scope of this work but could be the subject of future work.

Oberoi *et al.* [17] proposed *parallelism in the front-end* in which several instruction sequence fragments are fetched and renamed in parallel from a banked instruction cache. The focus of our work is to understand the energy-efficiency implications of sequential fetch organizations such as instruction cache and trace cache. Our work evaluates the IC with NTP fetch organization which is a simplified, sequential version of the Oberoi work. Our IC with NTP is evaluated to isolate the effects of implicit branch prediction of traces. Parallelized trace construction fetch organizations are beyond the scope of this work but are planned for future evaluation.

Several high fetch bandwidth mechanisms such as branch address cache [34], subgraph predictor [6], collapsing buffer [5], multiple-block ahead predictor [26], block-based trace cache [2] and trace cache [14, 19, 22, 24] have been proposed. Many of these mechanisms have drawbacks in terms of complexity. Therefore, for this work we only consider the trace cache described by Rotenberg [24] and the block-based trace cache described by Black *et al.* [2].

We are not aware of any further research which has examined the relative power-energy-performance tradeoff between fetch organizations which have only instruction caches and fetch organizations which have a combination of instruction cache and trace cache.

3 Simulation Techniques

All experiments in this work use SimpleScalar [4] and a modified Wattch [3] infrastructure with a power model based on the Alpha 21364 [29]. The base out-of-order simulator was extended to include CTC, STC, BBTC, and path-based next trace predictor (NTP) models. The microarchitecture model is summarized in Table 1.

To more closely study the efficiency of the fetch engines, we chose a highly parallelizing execution core. We altered the base microarchitecture to have 128 fetch queue entries, 128 register rename entries, and 128 load/store queue entries. In addition, we altered the base architecture so that as many as 16 instructions can be issued, executed, and committed in one cycle. Thus, a maximum of 16 IPC is possible with a perfect fetch engine and perfectly parallel code.

Since current CPU designs are increasingly using conditional clocking techniques to reduce power consumption, we calculate the power and energy metrics using Wattch's conditional clocking method which scales power linearly with port or unit usage [3]. To model leakage, when the port or unit is not in use, a fixed ratio of maximum power dissipation is charged:

Processor Core	
Active List	128 entries
Physical registers	80
LSQ	128 entries
Issue width	16 instructions per cycle
Functional Units	16 IntALU,4 IntMult/Div, 8 FPALU,4 FPMult/Div, 2 mem ports
Memory Hierarchy	
L1 D-cache Size	64 KB, 2-way, LRU, 64 B blocks, writeback
L1 I-cache Size	64 KB, 2-way, LRU, 64 B blocks both 2-cycle latency
L2	Unified, 4 MB, 8-way LRU, 128B blocks, 12-cycle latency, writeback
Memory	225 cycles (75ns)
TLB Size	128-entry, fully assoc., 30-cycle miss penalty
Branch Predictor	
Branch predictor	Hybrid PAg/GAg with GAg chooser
BTB	2 K-entry, 2-way
RAS	32-entry

Table 1: Simulated processor microarchitecture.

10% in most experiments. For the leakage experiments in Section 6, the leakage ratio is varied from 10% to 50% of maximum power dissipation.

4 Experimental Methodology

We conducted experiments to evaluate CTC, STC, BBTC, and IC fetch organizations. First, CTC, STC and BBTC fetch units were compared to IC with branch predictor (*IC-Bpred*) with and without area budget restrictions for the fetch engine components. In order to eliminate differences in trace prediction accuracy, IC with NTP (*IC-NTP*) and BBTC with NTP (*BBTC-NTP*) were also evaluated. Next, to eliminate the potential effects of improved branch prediction for some of the fetch organizations, we performed a set of experiments in which all fetch units' branch prediction accuracies were artificially equalized to that of the fetch engine with the best branch prediction accuracy (STC).

We evaluate the impact of increasing static power dissipation on fetch engine energy-efficiency by varying the leakage ratio from 10% to 50%.

We then present and evaluate a pipelined NTP to improve next trace prediction in the face of decreasing access times and shrinking structure areas.

4.1 CTC and STC Model

The STC modeled in the experiments is the one described by Rotenberg [23]. The sequential trace cache consists of an NTP [11] which predicts the next trace to be fetched, outstanding trace buffers (OTB) to hold in-flight predicted traces, and the trace cache itself. We modeled sequential trace cache access as described in the work of Hu *et al.* [9]. The instruction cache is only probed on the next cycle after a trace cache miss. Figure 1 shows the STC model. The sequential trace cache's power was modeled as an array structure, similar to an instruction cache, with one read and one write port. The conventional trace cache is basically the same fetch organization as the STC except that the instruction cache and the trace cache are probed in parallel. The power model for the CTC is adjusted to reflect the parallel trace cache and instruction cache access.

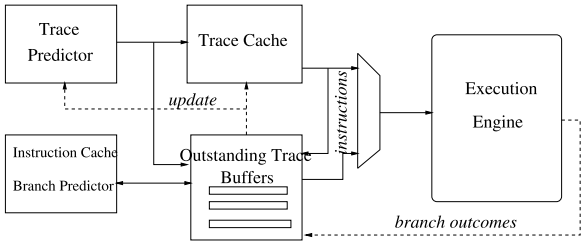


Figure 1: STC model (Patterned after figure in [23]).

Traces may be defined in many ways. Since we use the NTP of Jacobson *et al.* [11], we use the definition of trace used in their work for all CTC and STC simulations. A trace has a maximum of 16 instructions and as many as 7 branches (6 internal branches, plus a possible 7th terminating branch). Indirect branches terminate a trace. The NTP [11] uses path history information (recently committed traces) to make predictions much like a GAs or global history branch predictor. This information is combined with trace history to index a table that makes a prediction about the next trace to be fetched. In our experiments, 8 previous trace identifiers are hashed together to get indexes into the 64K-entry correlating table, and into the 32K-entry secondary table. A selector mechanism chooses the prediction from the more accurate table.

To model the power of the hybrid NTP [11], the correlating table, secondary table, return history stack (RHS) and path history register are each modeled as array structures with one read and one write port.

The outstanding trace buffer (OTB) maintains information about in-flight traces. When an entire trace commits, the trace is written to the trace cache (if needed) and the OTB entry is reclaimed. OTB entries also maintain information needed to recover from mispredicted branches. The power for the OTB is modeled as an array structure with two read ports and one write port. One read port is shared by fetch and mispredict recovery mechanisms and one read port is devoted to the commit time mechanism. The single write port is shared between fetch and mispredict recovery mechanisms. The experiments in Sections 5 and 5.2 use 128 OTB entries, while the experiments in Section 6 use 16 OTB entries based on a sensitivity study showing that an OTB with 16 entries does not incur significant performance loss.

4.2 BBTC Model

The BBTC described by Black *et al.* [2] modeled in our experiments consists of a trace table which makes next trace predictions, a block cache which stores basic blocks for trace construction, a rename table to maintain fetch address renaming and a fill unit which controls the update of the other three components. The trace table predicts a series of blocks to fetch using block-id execution history and branch history bits. These predicted blocks are fetched

from the block cache and assembled to construct a trace. Blocks are allocated to the block cache by the rename table which maintains a mapping of fetch addresses to block identifiers. The fill unit controls the update of the trace table, block cache and rename table.

Traces in the BBTC are defined to be a series of blocks with each block being defined as a series of instructions terminated by a branch, or a user-defined maximum number of sequential instructions. There are no other special trace termination conditions. To make the BBTC trace definition more comparable to the STC trace definition, we alter the BBTC trace definition to terminate traces on indirect branches as in the Rotenberg trace definition. Our experiments show that this modification in the trace definition improves the performance of the BBTC. We chose to model a replication of four and a maximum basic block size of six instruction to match the published best-performing BBTC. Figure 2 shows the BBTC model.

For power modeling, the BBTC components are each modeled as array structures. Each component of the BBTC has one read port and one write port.

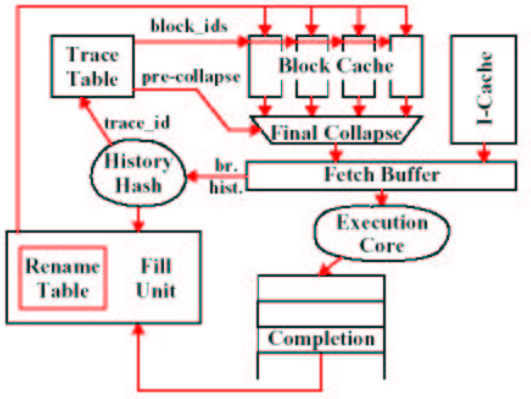


Figure 2: BBTC model (from [2]).

4.3 Cache Parameters

The fetch engine experiments which contain either CTC, STC or BBTC also include a non-interleaved instruction cache which serves as backup in the case of a trace cache miss. The fetch engine components which were held constant are shown in Table 2.

Component	Configuration
I-cache	512 set, 64B line, 2-way, LRU
Branch predictor	Hybrid: 4K-entry PAg, 4K-entry GAg (12-bit history) 4k-entry GAg chooser 2k-entry, 2-way set associative BTB 32-entry RAS
OTB	128 entries
NTP	64K-entry correlating table 32K-entry secondary table 128-entry RHS

Table 2: Parameters held constant for STC and BBTC experiments.

4.3.1 CTC and STC Configurations

Since the number of components in the CTC, STC and BBTC designs differ from the number of components in IC designs, an equal-area comparison is difficult. Therefore, we examine the fetch engines over a range of different fetch engine areas. We first evaluate the fetch engines when the area of individual fetch engine components is unrestricted. This allows us to examine the theoretical potential of the various fetch engines. Then, to consider access time, we restrict the area of each individual component of each fetch engine to areas of 2 KB through 512 KB in successive simulations.

Associativities for the STC are varied in the experiments but the replacement policy is fixed to LRU, and the line size is fixed at the length of one trace. Table 3 shows the fetch engine areas used in the experiments of Section 5. These experiments use the ideal NTP and OTB parameters specified by [11]. The area used for the CTC/STC alone is listed alongside the total fetch engine area. The remaining fetch engine area is calculated by totaling the area of the backing instruction cache, branch predictor (including BTB), OTB, and hybrid NTP.

In the first comparison, the area of the STC is varied while the areas of the other components are held constant (See Table 2). In a second comparison, the areas of the STC, NTP, and OTB are limited to 2 KB through 512 KB.

Fetch Engine Area	CTC/STC Area	Fetch Engine Area	IC Area
980 KB	16 KB	100 KB	64 KB
996 KB	32 KB	164 KB	128 KB
1028 KB	64 KB	292 KB	256 KB
1092 KB	128 KB	548 KB	512 KB
1220 KB	256 KB	1060 KB	1024 KB
1476 KB	512 KB		

Table 3: Fetch engine area and corresponding CTC/STC and IC areas used in experiments which use default NTP and OTB components. Cache area is included in the fetch engine area total.

4.3.2 BBTC Configurations

Similarly, the associativities of the BBTC are varied. Fetch engine and component areas for BBTC with trace table and BBTC with NTP are summarized in Tables 4 and 5.

Fetch Engine Area	Trace Table Area	Rename Table Area	Block Cache Area
268 KB	32 KB	8 KB	128 KB
436 KB	64 KB	16 KB	256 KB
772 KB	128 KB	32 KB	512 KB
1444 KB	256 KB	64 KB	1024 KB

Table 4: Fetch engine area and corresponding BBTC component areas used in BBTC with trace table experiments.

In experiments which model an IC-only fetch engine, we use a 2-way set associative, 2-

Fetch Engine Area	Rename Table Area	Block Cache Area	NTP Area
1746 KB	8 KB	128 KB	1510 KB
1930 KB	16 KB	256 KB	1558 KB
2250 KB	32 KB	512 KB	1606 KB
2842 KB	64 KB	1024 KB	1654 KB

Table 5: Fetch engine area and corresponding BBTC component and NTP areas used in experiments which use default NTP and OTB components. Note that NTP area varies as size of block cache index/area varies and that no trace table is included.

way interleaved instruction cache with 64 byte lines, and LRU replacement. For the *IC-Bpred* unrestricted component area experiments, the fetch engine area is comprised of the area for the IC and the branch predictor, with the branch predictor area held constant. For the *IC-NTP* experiments, a fixed-area NTP with a 2K-entry, 2-way set associative BTB is used without a backing branch predictor. Therefore the *IC-NTP* fetch engine area is comprised of the areas of the correlating table, secondary table, RHS, BTB, and IC. These parameters are listed in Table 2. The IC areas and the area of the entire fetch engine are listed in Table 3.

4.4 Benchmarks

We evaluate our results using benchmarks from the SPEC CPU2000 suite. The benchmarks are compiled and statically linked for the Alpha instruction set using the Compaq Alpha compiler with SPEC *peak* settings and include all linked libraries but no operating-system or multiprogrammed behavior. Seven integer benchmarks (*gzip, gcc, crafty, parser, eon, perlbmk,* and *vortex*) and five floating point benchmarks (*wupwise, mesa, art, facerec*, and *ammp*) were used in the experiments.

Our initial experiments demonstrated little performance benefit from larger fetch engines on the floating point benchmarks. We suspect that this is because they have a small text size and are highly predictable. Thus, results for floating point benchmarks are not shown and can be found in [33].

Benchmark	Input	Fastforward (insts)
164.gzip	ref graphic	77.3 B
176.gcc	ref expr	1.3 B
186.crafty	ref	72.8 B
197.parser	ref	183.8 B
252.eon	ref rushmeier	36.3 B
253.perlbmk	ref diffmail	13.3 B
255.vortex	ref lendian3	28.3 B

Table 6: Fastforward numbers for benchmarks. Benchmarks are fastforwarded and then warmed up for 300 M instructions before statistics gathering.

Simulations are fast-forwarded according to the numbers in Table 6 [27], then run in full-detail cycle-accurate mode (without statistics-gathering) for 300 million instructions to train the caches—including the L2 cache—and the branch predictor before statistics gathering is started. This interval was found to be sufficient to yield representative results [10].

The individual results for each benchmark exhibited similar trends, therefore our results are presented as the average of the benchmarks.

5 Fetch Engine Area Exploration

We performed a comparison of the following fetch engine designs: CTC with NTP (*CTC-NTP*), STC with NTP (*STC-NTP*) [11], BBTC with trace table (*BBTC-TT*), and IC with branch predictor (*IC-Bpred*). To eliminate the effects of enhanced next trace prediction, IC with NTP (*IC-NTP*) and BBTC with NTP (*BBTC-NTP*) are also evaluated. The areas listed in Tables 3, 4, and 5 were used. STC and BBTC associativity and area were varied and the IPC and energy-delay-squared (ED^2) were analyzed. We choose to examine ED^2 as a metric because it considers both power dissipation and performance and is voltage independent.

Increased associativity improved the IPC for CTC, STC, and BBTC, but showed only modest improvement in ED^2. We present direct-mapped results for the fetch engines because it represents the worst performing associativity and ED^2.

5.1 Unrestricted Component Area

When fetch engine components were not restricted to a specific area budget, we found that for approximately equal fetch engine area, an STC design has better performance and comparable ED^2 relative to *IC-Bpred* and *IC-NTP*. This is shown in Figure $4(a)$. The STC configurations outperform (in terms of IPC) the best-performing *IC-Bpred* and *IC-NTP* configurations by a maximum of 11.3% and 5.3% respectively. The results also show that an STC fetch engine generally has better energy-efficiency with the exception of the 100 KB and 164 KB *IC-Bpred* areas. A 231KB STC fetch unit has 9.0% lower ED^2 than a 292 KB *IC-Bpred* fetch engine and 0.4% higher ED^2 than a 164 KB *IC-Bpred* configuration. A 231 KB STC fetch engine contains a 16 KB STC and a 64 KB instruction cache. The smaller STC is accessed roughly two-thirds of the execution time and the larger area instruction cache is accessed the remaining time. Accessing a smaller area saves energy due to smaller row and column decoders. This differs from the *IC-Bpred* and *IC-NTP* fetch units which must rely solely on the instruction cache. Another reason that might explain the better performance of the STC is improved branch prediction from the NTP. This is explored in the Section 5.2.

CTC performs approximately the same as STC but with higher ED^2 due to the parallel IC and trace cache access.

The *BBTC-TT* configurations have the lowest IPC (on average 16% lower compared to STC) and highest ED^2 (on average 27% higher compared to STC) of the fetch engine designs. We believe that the low IPC is a result of poor next trace prediction from the trace table. This does not conflict with the Black's [2] results whose work simulates perfect trace prediction to explore the performance potential of the BBTC. The results for *BBTC-NTP*, which has improved branch prediction accuracy, exhibit IPCs close to the STC configurations (within 18%) but with higher ED^2 (on average 17% higher) due to its greater area. For the remainder of this paper, we exclude *BBTC-TT* results due to its poor performance due to poor next trace prediction.

5.2 Eliminating Branch Prediction Effects

The STC's better performance in the initial experiments could be a result of the improvement in branch prediction accuracy provided by the NTP. We explored this possibility by performing two additional sets of experiments. In one set of experiments, we artificially equalized the branch prediction accuracy (per benchmark) of all the configurations to that of the average best performing STC configuration (*best*, 343KB, 4-way). In the second set of experiments, we equalized the branch prediction accuracy (per benchmark) to that of the worst performing STC configuration (*worst*, 231KB, 1-way). The results of these two sets of experiments showed that

equalizing branch prediction accuracy to *best* or *worst* did not affect the relationship between the various fetch engine designs, so only the results for *best* are shown in Figure 4.

The IPC of the *IC-Bpred* fetch engines improved 3.3% on average compared to the initial experiments. Its ED^2 was roughly 10.1% lower than without branch prediction equalization. This shows that *IC-Bpred* designs can benefit significantly from the artificial branch prediction equalization. The performance of the STC fetch engines is decreased in *worst* because the branch prediction equalization process artificially forces more branch mispredictions than the STC fetch engine would normally make. This results in an ED^2 increase due to extra cycles spent recovering from additional mispredictions. Conversely, *IC-Bpred* and *IC-NTP* designs exhibit an improvement in IPC and a decrease in ED^2 because their branch prediction accuracy is artifically improved by the branch prediction equalization technique.

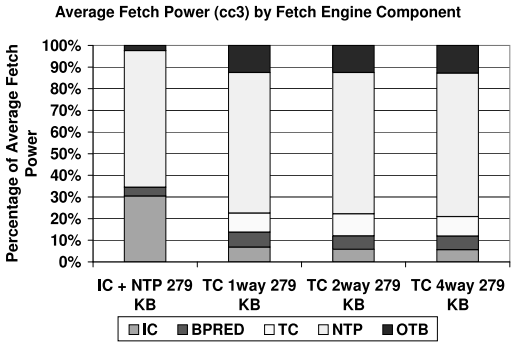


Figure 3: Percentage of average fetch power consumed by fetch engine components (results for equalization to worst trace cache configuration)

With branch prediction equalization, *IC-NTP* achieves IPC similar to that of STC fetch engines (within 0.8%) and similar ED^2 to STC. Its ED^2 slope is very similar to that of *IC-Bpred* but slightly higher due to the increased power consumption from accessing the large NTP and OTB components (Figure 3). These results indicate that an IC fetch engine design cannot attain the IPC and energy-efficiency of STC designs by simply replacing the branch predictor with an NTP. A fetch unit consisting of an IC, and NTP backed by a branch predictor might attain similar performance to STC fetch engines, but likely with increased area and fetch power.

5.3 Summary Results

These experiments show that without considering the effects of improved branch prediction, STC fetch engine designs can achieve a significant performance improvement (5.3%) at an ED^2 similar to *IC-Bpred* fetch engines. When artificially equalizing for branch prediction effects, STC fetch engines have 1.6% lower IPC and 18.9% higher ED^2 (279KB 4-way STC vs. 164 KB IC). We also see that for equal area components *IC-NTP* is very comparable to STC. However, for *IC-Bpred* fetch engines to attain this degree of ED^2 improvement, improved branch prediction mechanisms must be found. These experiments are only intended to isolate the contribution of branch prediction from instruction storage.

5.4 Restricted Component Area

Clock rates in modern processors are increasing rapidly. As a result, the time delay to access structures is more important than ever. To account for access time considerations, we performed an experiment where the area of each component of each fetch engine was limited to a fixed area budgets ranging from 2 KB to 512 KB. We perform this experiment with all fetch engines having direct-mapped structures where applicable. For example, for BBTC sim-

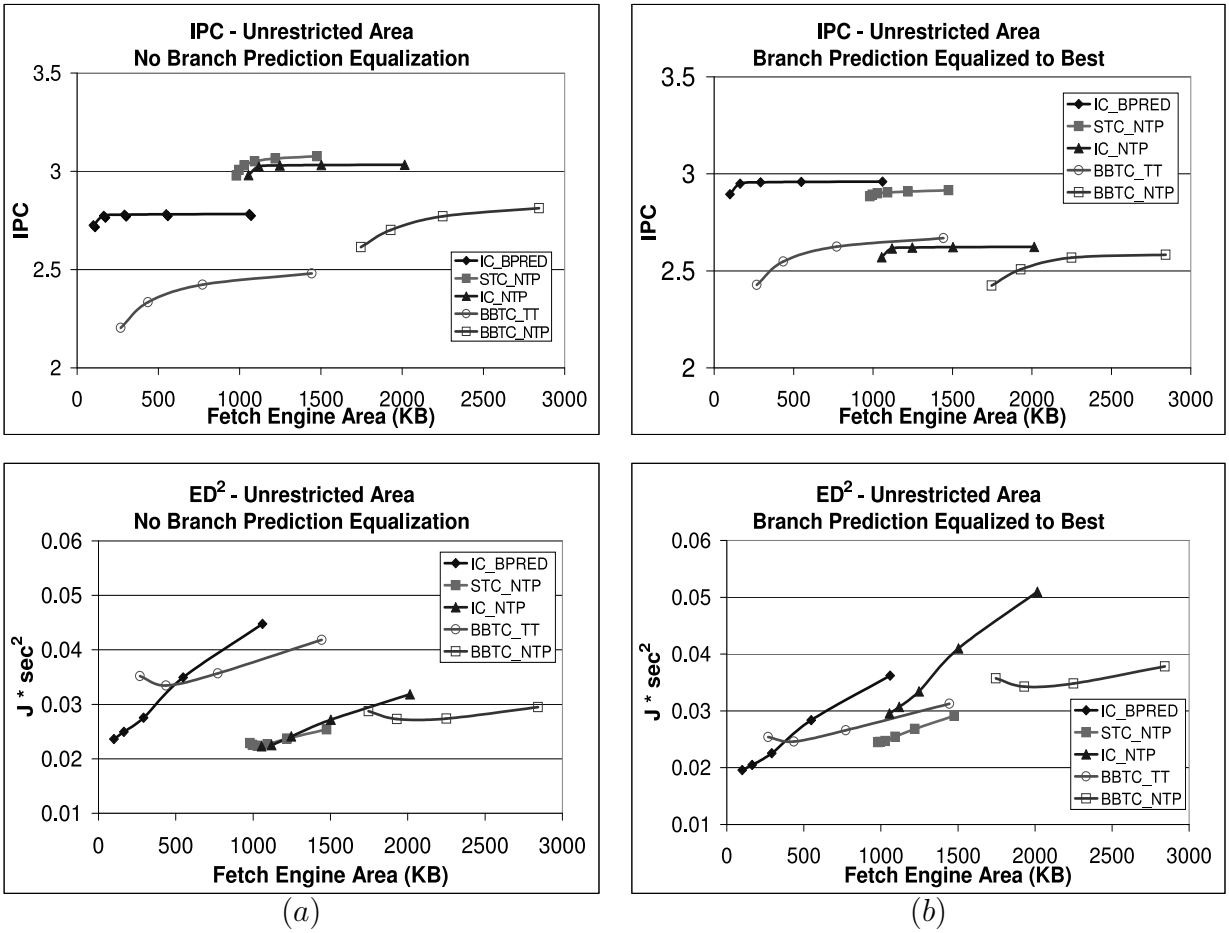


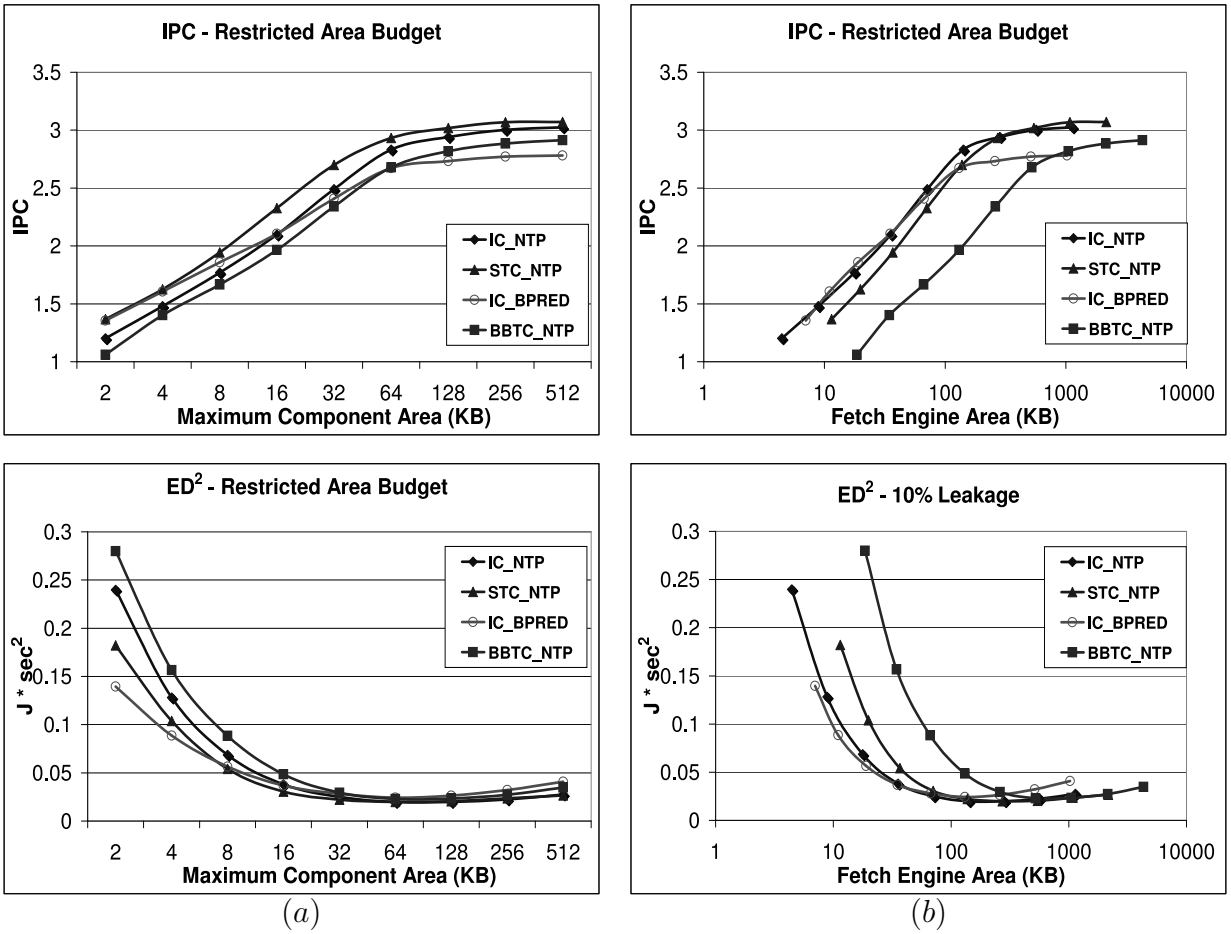
Figure 4: IPC and ED^2 when branch prediction rates are (a) unadjusted and (b) equalized to best performing STC configuration (Area of components unrestricted)

ulations, the area of each of the BBTC components (trace table, rename table, block cache), backing instruction cache and branch predictor was limited to a fixed area. We also examined the effect of increasing leakage. Results are shown in Figure 5.

STC gets higher IPC than the other fetch engines starting at 32 KB component areas, followed closely by *IC-NTP*, *IC-Bpred*, and *BBTC-NTP*. Figure 6 shows that STC does better than *IC-Bpred* in terms of IPC even at smaller areas. However, STC does not show an ED^2 improvement over *IC-Bpred* until 16 KB where the overhead of the extra area of the STC begins to pay off.

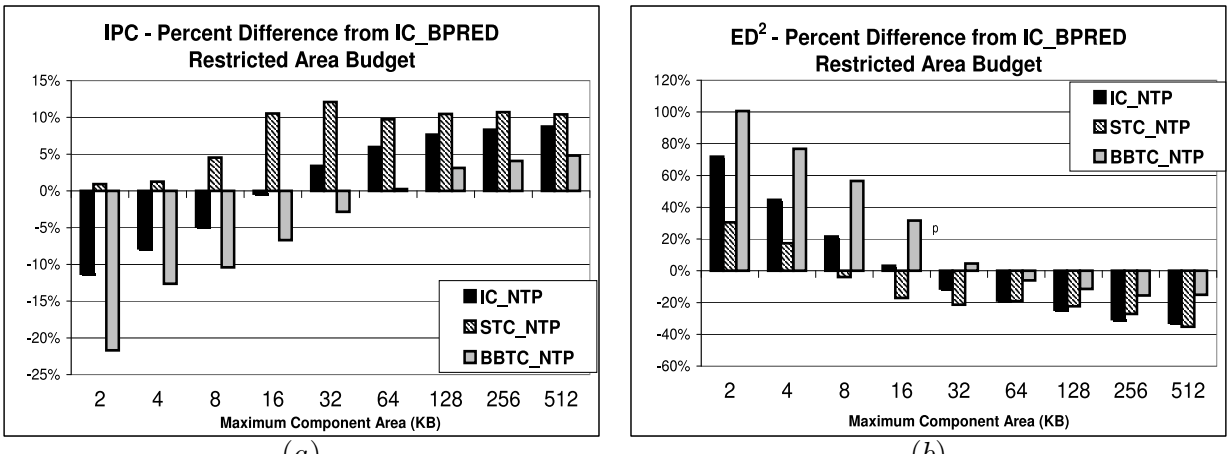
When the components of each fetch engine are restricted to a specific area budget the effect of leakage is decreased (not a significant effect) compared to the results when fetch engine area budget is unrestricted. Each component in each fetch engine is no larger than a fixed area, so naturally the effect of leakage for each fetch engine is reduced, but the performance of certain structures might be limited by the area restrictions.

For the BBTC the published best configuration has an 8k-entry trace table (128KB) and 4k-entry block cache (512 KB). The block cache (the main component of the BBTC) may be penalized under the area restriction. This explains why the BBTC does not fare as well for the restricted component area experiments.



(a) (b)

Figure 5: Equal area fetch components when branch prediction rates are unadjusted: (a) by maximum component area and (b) by total fetch engine area



(a) (b)

Figure 6: Percent difference of (a) IPC and (b) ED^2 relative to *IC-Bpred*

6 Leakage Current Sensitivity

Total power dissipation due to chip leakage is projected to exceed total dynamic power as feature sizes reach 65 nm [28]. To ensure that we consider the energy-efficiency of CTCs, STCs, and ICs both now, and in future process technologies, we examine the results of varying the leakage ratio from 10% to 50% of maximum power dissipation (Figure 7).

We find that when component areas are restricted to account for access delay, increased leakage ratio has little effect. This is due to the fact that the components for the respective fetch organizations are relatively equal in size. [1]

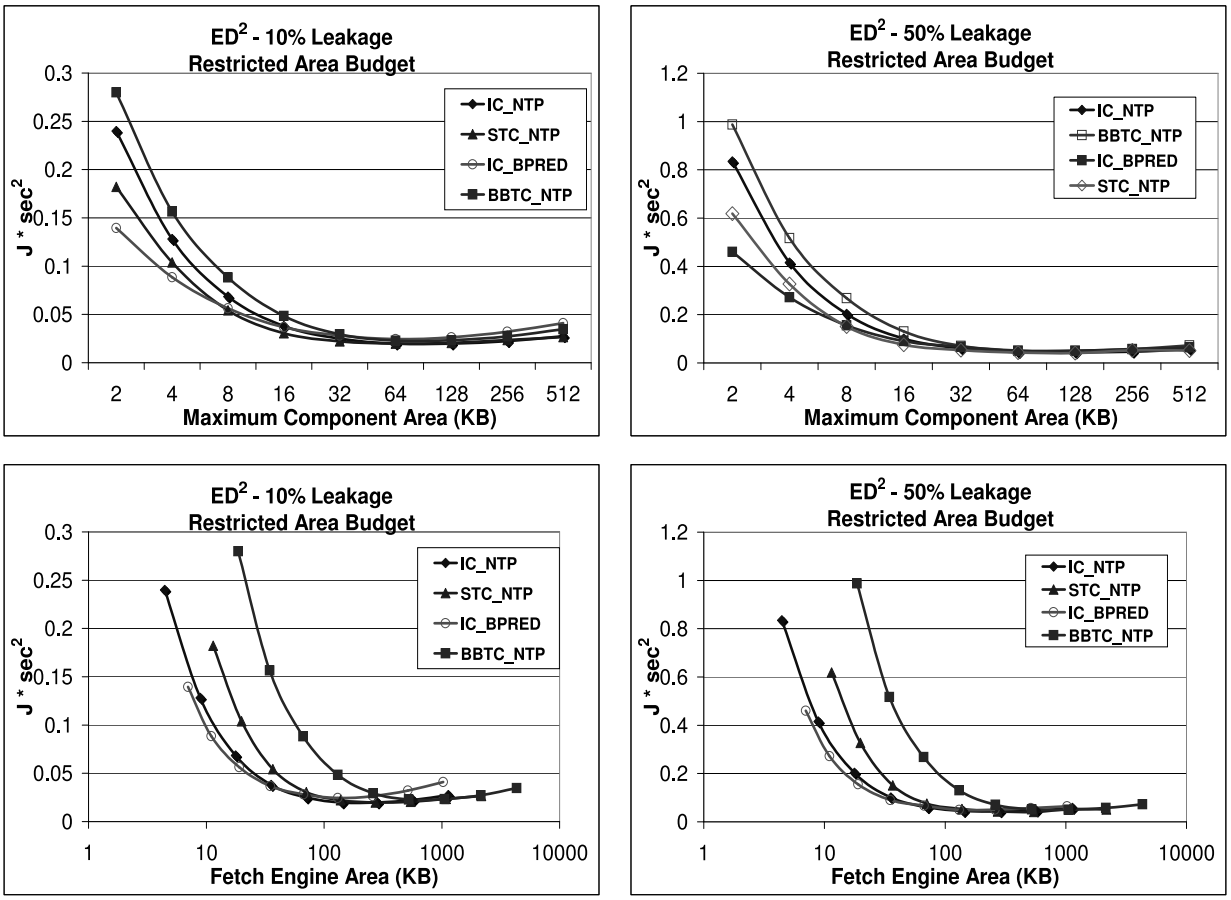


Figure 7: ED^2 at leakage ratios: (a) 10% and (b) 50% (top row plotted by maximum component area, bottom row plotted by total fetch engine area)

7 Ahead Pipelining the NTP

Faster clock rates leads to shorter cycle times. Shorter cycle times makes accessing larger structures more challenging. Our experimental results show that branch prediction accuracy is an important factor in fetch engine performance. Our results show that the NTP provides better branch prediction accuracy than a hybrid branch predictor. However, at very small component areas, the NTP performs poorly. Ahead-pipelining is one technique to enable a structure to be larger and still be able to produce output each cycle.

[1] We do not present separate IPC graphs in this experiment because the leakage ratio has no effect on IPC. Refer to Figure 4 for IPC results.

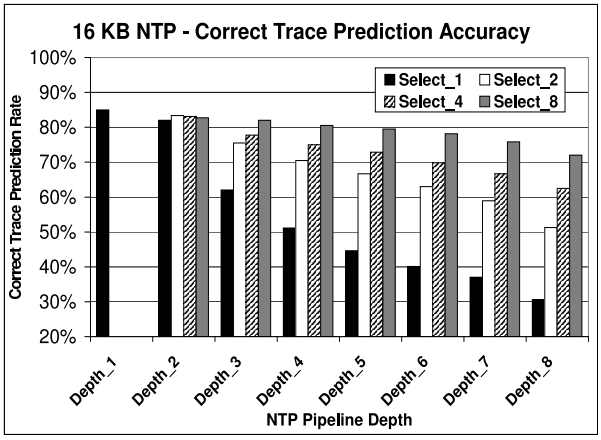


Figure 8: NTP correct trace prediction accuracy as affected by pipeline depth and number of entries selected for final prediction.

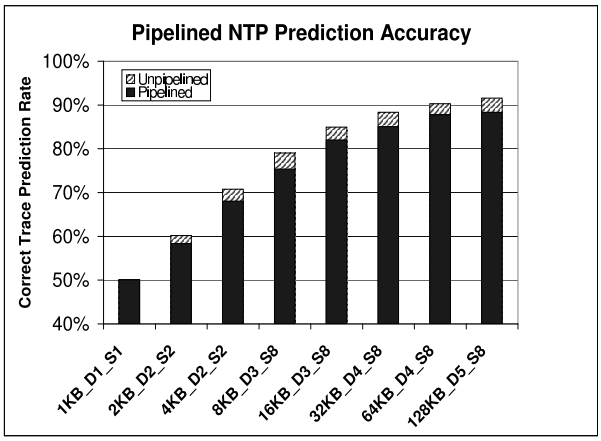


Figure 9: Improvement of NTP correct trace prediction accuracy as NTP table areas increase. (D = NTP pipeline depth, S = number of entries selected for consideration in final trace prediction)

Ahead pipelining initiates a prediction many cycles in advance of when it is needed to hide access latencies. To do so, older information must be used to generate a set of the predictions for selection and at the last moment the most current information is used to select the final prediction. Patt [34] evaluates pipelined access to a branch address cache to perform multiple branch prediction. Jimenez [12], Seznec [26, 25] and Tarjan *et al.* [32] evaluate ahead-pipelining single-branch predictors in order to get better prediction accuracy (enable larger branch predictor structures) while considering the impact of delay. We apply the ahead-pipelining concept to the NTP trace prediction mechanism and perform a set of experiments in which we compare the performance and energy-efficiency of fetch engines with the pipelined NTP.

Ahead-pipelining the NTP is a way to reduce some of the performance loss from reduced area due to cycle time restrictions. It allows us to plan structures larger than can be accessed within a single cycle, and yet still produce accurate output each cycle.

Roughly 1KB can be accessed within a single cycle [13]. If a structure can be pipelined to 2 stages, the structure could be made as large as 4 KB. As the depth of the pipeline increases,

the area for a particular structure that can be accessed roughly quadruples [12].

We pipeline the NTP by using incomplete trace history to select a range in the NTP tables and in the cycle before the prediction is needed, use the most current trace history to select the actual prediction to be made. This technique is similar to the technique used by Patt [34] and Jimenez [12] except that we apply it to next trace prediction which is a form of implicit multiple branch prediction. Trace misprediction latency is modeled as the depth of the NTP pipelining (flushing the NTP pipeline).

We compare 1KB - 32KB component areas, assuming that the NTP pipeline depth must increase as we increase the area. We vary the pipeline depth from 1 (non-pipelined) to 6 and vary the range of entries chosen by the incomplete history from 1,2,4,8. (See Table 7)

Component Area	Pipeline Depth
1 KB	1
2 KB	2
4 KB	3
8 KB	4
16 KB	5
32 KB	6

Table 7: NTP component areas and corresponding NTP pipeline depth. Number of entries selected in advance is varied at values 1,2,4, and 8.

Figure 8 shows the NTP correct trace prediction accuracy of the NTP for varying pipeline depths and selection ranges for the 16 KB component area. (All structures in the NTP can be no larger than 16 KB). We choose to show the NTP trace prediction accuracy in order to demonstrate the potential for pipelining the NTP. The bar labeled *Depth1* represents the maximum attainable NTP prediction accuracy since it is the single-cycle trace prediction accuracy. As the depth of the pipeline increases, if only a single entry is selected in the early stages, the trace prediction accuracy rapidly decreases. This is due to the use of only older history and no newer history to make the next trace prediction. Increasing the range of entries to select from at prediction time improves the prediction accuracy. As the number of entries selected is increased to 8, the difference in trace prediction accuracy from non-pipelined NTP rapidly decreases. As the depth of the pipeline increases, the trace prediction declines due to the use of more and more old history and less new history to make the trace prediction.

Figure 9 compares the NTP trace prediction accuracy of comparable pipelined NTP design points. A non-pipelined 1KB area NTP can be compared to a progressively larger, more deeply pipelined NTP. The dark colored section of the bar represents the trace prediction accuracy when pipelined, while the shaded area of the bar represents the trace prediction accuracy if the same area structure were to be accessible within a single cycle. These results show that the trace prediction accuracy does not suffer too much of a penalty from being pipelined.

Figure 6 shows that a 16KB STC accessible in a single cycle shows significant ED^2 improvement over *IC-Bpred*. Since this is not accessible in a single cycle, we compare the performance of our 16KB 2-deep pipelined *STC-NTP* with a 2 KB non-pipelined

STC-NTP and find that we get a 26.7% improvement in IPC and 25.8% reduction in ED^2. This suggests that with ahead pipelining, STCs can provide significant performance benefits and increased energy-efficiency.

8 Conclusions

We implement and verify the performance of the STC model proposed by Rotenberg [24] and the BBTC model proposed by Black [2] and augment both models to include power and energy modeling. Our experiments show that when fetch components are not constrained by access time, fetch engines which include STCs are more energy-efficient while providing a significant performance improvement over IC-only fetch engine organizations. Even when branch prediction accuracy is artificially equalized, STC fetch engines are still as energy-efficient as ICs. The ED^2 results show that although an STC and its supporting components may take up more chip area than an IC-only fetch engine, it yields better energy-efficiency overall (without branch prediction equalization) due to better opportunities for accessing smaller area fetch engine components. These results represent the theoretical benefit of trace caches, which stem partly from the implicit multiple branch prediction of the NTP and partly from the benefit of storing instructions as traces as opposed to static program blocks.

We then consider the trend of increasing access delay due to higher clock speeds by limiting the area of the fetch engine structures and consider the effect of increasing leakage ratio. We find that the benefit of STC fetch engines compared to IC fetch engines in the face of increasing delay is more modest than when delay is not considered.

We introduce and evaluate an ahead pipelined NTP to address the trend of increasing access delay. We find that there is a modest decrease in *STC-NTP* performance over the non-pipelined area equivalent. When comparing to a single-cycle accessible non-pipelined 2 KB *STC-NTP*, a 2-deep pipelined 16 KB *STC-NTP* (3 selection bits) can provide a 26.7% IPC improvement and 25.8% ED^2 improvement.

Future directions for this work include exploring the energy-efficiency of other fetch engine designs including instruction streams [21], dynamic prediction directed trace cache [8], the filter trace cache [31], and parallel fetch designs [17].

Acknowledgments

This work is supported in part by the National Science Foundation under grant nos. CCR-0133634, CCR-0105626, and EIA-0224434, and a grant from Intel MRL. We would also like to thank Eric Rotenberg, Dee A.B. Weikle, and Jason D. Hiser for their helpful input.

References

[1] R. Iris Bahar, G. Albera, and S. Manne. Power and performance tradeoffs using various caching strategies. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, pages 64–69. ACM Press, 1998.

[2] B. Black, B. Rychlik, and J.P. Shen. The block-based trace cache. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 196–207. IEEE Computer Society Press, May 1999.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[4] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.

[5] T. Conte, K. Menezes, P. Mills, and B. Patel. Optimization of instruction fetch mechanisms for high issue rates. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 333–344, Oct 1996.

[6] S. Dutta and M. Franklin. Control flow prediction with tree-like subgraphs for superscalar processors. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 258–263, Nov 1995.

[7] D.H. Friendly, S.J. Patel, and Y.N. Patt. Alternative fetch and issue policies for the trace cache fetch mechanism. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 24–33, 1997.

[8] J.S. Hu, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir. Using dynamic branch behavior for power-efficient instruction fetch. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2003)*, 2003.

[9] J.S. Hu, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Power-efficient trace caches. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE '02)*, 2002.

[10] J. W. Haskins, Jr. and K. Skadron. Memory reference reuse latency: Accelerated sampled microarchit ecture simulation. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Syste ms and Software*, pages 195–203, Mar. 2003.

[11] Q. Jacobson, E. Rotenberg, and J. E. Smith. Path-based next trace prediction. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 14–23, 1997.

[12] D. Jimenez. Reconsidering complex branch predictors. In *In Proceedings of the 9th International Symposium on High Performance Computer Architecture*, pages 43–52, 2003.

[13] D. A. Jimenez, S. W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 67–76. ACM Press, 2000.

[14] J. Johnson. Expansion caches for superscalar processors. Technical Report CSL-TR-94-630, Computer Science Laboratory, Stanford University, June 1994.

[15] N.S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 219–230. IEEE Computer Society Press, 2002.

[16] J. Montanaro and et al. A 160-mhz 32-b 0.5-w cmos risc processor. Technical Report Vol. 9, Digital Technical Journal, Digital Equipment Corporation, 1997.

[17] P.S. Oberoi and G.S. Sohi. Parallelism in the front-end. In *Proceedings of the 30th annual international symposium on Computer architecture*, pages 230–240. ACM Press, 2003.

[18] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 233–44, Feb. 2002.

[19] A. Peleg and U. Weiser. Dynamic flow instruction cache memory organized around trace segments independent of virtual address line. In *U.S. Patent Number 5,381,533*, Jan. 1995.

[20] R. Rakvic, B. Black, and J.P. Shen. Completion time multiple branch prediction for enhancing trace cache performance. In *Proceedings of the 27th annual international symposium on Computer architecture*, pages 47–58. ACM Press, 2000.

[21] A. Ramirez, O.J. Santana, J.L. Larriba-Pey, and M. Valero. Fetching instruction streams. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 371–382. IEEE Computer Society Press, 2002.

[22] E. Rotenberg, S. Bennett, and J. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching. Technical Report 1310, Cs Dept. University of Wisconsin, Madison, April 1996.

[23] E. Rotenberg, S. Bennett, and J. E. Smith. A trace cache microarchitecture and evaluation. *IEEE Transactions on Computers*, 48(2):111–120, 1999.

[24] E. Rotenberg, S. Bennett, and J.E. Smith. Trace cache: A low latency approach to high bandwidth instruction fetching. In *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 24–35, 1996.

[25] A. Seznec. Revisiting the Perceptron Predictor. Technical Report 1620, IRISA, May 2004.

[26] A. Seznec, S. Jourdan, P. Sainrat, and P. Michaud. Multiple block ahead branch predictors. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct 1996.

[27] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.

[28] SIA. *International Technology Roadmap for Semiconductors*, 2001.

[29] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 2–13, June 2003.

[30] B. Solomon, A. Mendelson, D. Orenstein, Y. Almog, and R. Ronen. Micro-operation cache: a power aware frontend for the variable instruction length isa. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 4–9. ACM Press, 2001.

[31] W. Tang, R. Gupta, and A. Nicolau. Design of a predictive filter cache for energy savings in high performance processor architectures. In *Proceedings of the 2001 International Conference on Computer Design*, pages 68–73, 2001.

[32] D. Tarjan, K. Skadron, and M. Stan. An ahead pipelined alloyed perceptron with single cycle access time. In *Proceedings of the 5th Workshop on Complexity-Effective Design*, 2004.

[33] Withheld. Withheld. Technical Report Withheld, Withheld, October 2003.

[34] T.-Y. Yeh, D.T. Marr, and Y.N. Patt. Increasing the instruction fetch rate via multiple branch prediction and a branch address cache. In *Proceedings of the 7th International Conference on Supercomputing*, pages 67–76, July 1993.

[35] Y. Zhang and J. Yang. Low cost instruction cache designs for tag comparison elimination. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pages 266–269. ACM Press, 2003.