Energy-Efficient Trace Reuse Cache for Embedded Processors

Yi-Ying Tsai and Chung-Ho Chen, *Member, IEEE*

***Abstract*—For an embedded processor, the efficiency of instruction delivery has attracted much attention since instruction cache accesses consume a great portion of the whole processor power dissipation. In this paper, we propose a memory structure called *Trace Reuse (TR) Cache* to serve as an alternative source for instruction delivery. Through an effective scheme to reuse the retired instructions from the pipeline back-end of a processor, the TR cache presents improvement both in performance and power efficiency. Experimental results show that a 2048-entry TR cache is able to provide 75% energy saving for an instruction cache of 16 kB, at the same time boost the IPC up to 21%. The scalability of the TR cache is also demonstrated with the estimated area usage and energy-delay product. The results of our evaluation indicate that the TR cache outperforms the traditional filter cache under all configurations of the reduced cache sizes. The TR cache exhibits strong tolerance to the IPC degradation induced by smaller instruction caches, thus makes it an ideal design option for the cases of trading cache size for better energy and area efficiency.**

***Index Terms*—Cache memories, computer architecture, energy management, microprocessors.**

I. Introduction

IMPROVING the efficiency of instruction delivery has been an important strategy in boosting processor performance. In addition to employ cache memories, schemes for control flow speculation are also proposed. Well-known research topics of this kind include branch prediction [1]–[6], instruction cache restructuring [7]–[9], and trace caches [10]–[13]. For the relatively simple and short pipeline of an embedded processor, incorporating a complex speculation scheme for instruction delivery is seldom considered an option in the past. On the other hand, simply allocating more hardware budget to increase the size of instruction cache has become a viable solution for embedded processors due to the advance of process technology. However, this also implies that the power dissipation ratio of the instruction cache has become increasingly more dominant in the total processor power usage. For instance, about 27% of the processor power of the StrongARM is dissipated in the instruction cache [14]. To achieve better energy efficiency for the cache system, the filter cache scheme [15], [16] has been proposed to trade performance for better energy efficiency of the cache system. The basic idea of the filter cache is to insert a tiny level-zero cache to provide the most frequently accessed data with lower energy expenses. However, this also induces performance degradation due to the increased cache access latency. Several follow-up researches have proposed various prediction techniques [17]–[20] to mitigate the performance degradation of the filter cache.

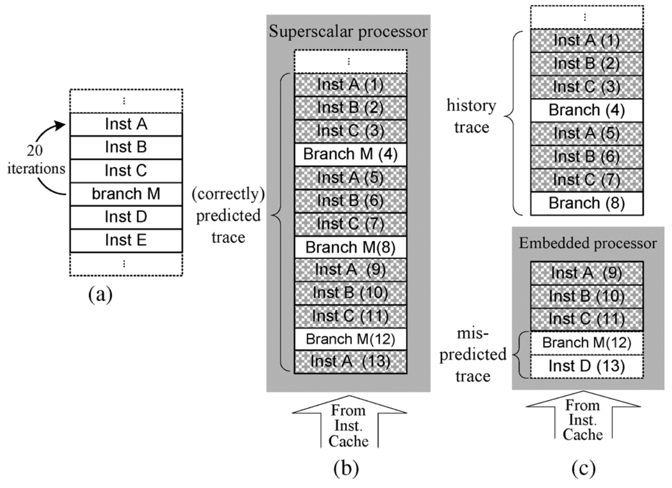

Fig. 1. Dynamic traces of a 20-iteration loop. (a) Static code layout; (b) processor with aggressive branch predictor; (c) processor with non-taken predictor.

Essentially, the prior works have focused on the front-end of the processor to improve either the performance or the energy efficiency of instruction delivery. In other words, all these efforts aim to speculate the right program traces prior to the branch instructions are resolved, and then manage to reduce the program execution latency or energy consumption via the speculated trace information. Since the speculated traces, given that they are correctly predicted, will ultimately be retired from the pipeline and become the history traces, these executed traces are potentially very beneficial for reuse in the case of an embedded processor. Fig. 1(a) shows a code sequence including a 20-iteration loop; Fig. 1(b) and (c) illustrate the processor contexts when the code in (a) is executed by two processors of different complexity. With the support of an aggressive branch predictor, the superscalar processor in (b) has buffered a long trace in its instruction window to exploit instruction level parallelism. For the embedded processor in (c), which simply predicts non-taken for branches, the unrolled loop has induced a branch misprediction for each iteration except the last one. Apparently the embedded processor will take more cycles to complete the loop

Manuscript received November 01, 2009; revised March 02, 2010 and May 18, 2010; accepted June 07, 2010. This work was supported in part by the National Science Council, Taiwan, under NSC Contract 96-2221-E-006-192-MY3 and 97-2221-E-006-250-MY3.

Y.-Y. Tsai is with the Electrical Engineering Department, National Cheng-Kung University, Tainan City 701, Taiwan (e-mail: magi@casmail.ee.ncku.edu.tw).

C.-H. Chen is with the Institute of Computer and Communication Engineering, Electrical Engineering Department, National Cheng-Kung University, Tainan City 701, Taiwan (e-mail: chchen@mail.ncku.edu.tw).

Digital Object Identifier 10.1109/TVLSI.2010.2055908

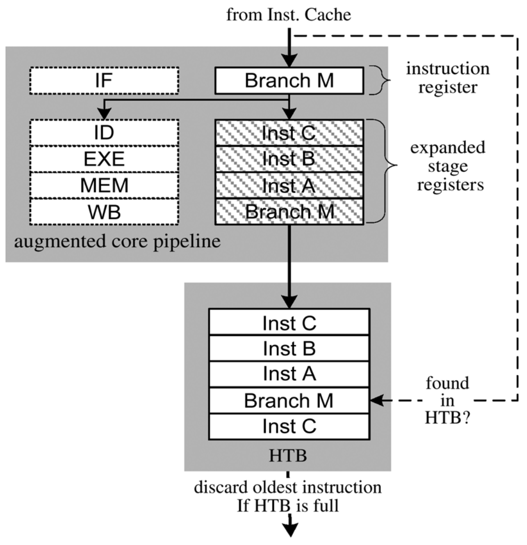

Fig. 2. Modified pipeline and the history trace buffer.

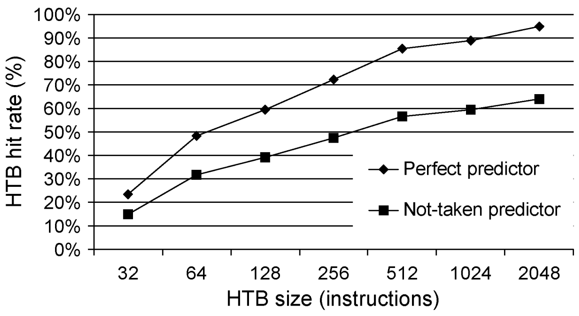

Fig. 3. Raw HTB hit rate of processor with different predictor.

due to branch misprediction penalty; yet it produces the identical history traces as those in processor (b). That is, the unrolled loop trace constructed by the sophisticated front-end of the superscalar processor is identical to the one collected from the pipeline back-end of the simpler embedded processor. Moreover, if the embedded processor were able to fetch instructions with the sequence presented in the history trace, future branch mispredictions could be avoided.

To investigate the feasibility of delivering instructions from the pipeline back-end, we perform simulations using the processor model depicted in Fig. 2. The architecture shown in Fig. 2 consists of an embedded processor with additional D flip-flops augmented at each stage of the pipeline and a history trace buffer (HTB) appended at the back-end. The augmented D flip-flops can be deemed as the expansions of the stage register, which help to preserve and pass the undecoded instruction bits to the back-end. The HTB is managed as a first-input-first-output (FIFO) buffer to capture a fixed length of the most recently retired instruction sequence.

We use the same simulation environment described in Section III to gather the statistics presented below. For each instruction fetched from the front-end, the HTB is searched to see if the same instruction also hits in the buffered history trace. The HTB hit counts and the total fetched instruction numbers are summed respectively throughout the simulation to calculate the *raw HTB hit rate*. The raw HTB hit rate indicates how often the opportunity occurs for the fetch logic to utilize HTB instead of the cache as the instruction source. The equation for raw HTB hit rate is H_k/F_k, where H_k is the HTB hit count and F_k is the total fetched instruction number for the $K_{\rm th}$ program. Obviously the total fetched instruction number F_k will be affected by the accuracy of the branch predictor employed at the fetch stage. The more precise the predictor is, the fewer the misfetched instructions are. Consequently, using a more accurate predictor results in a larger raw HTB hit rate due to the smaller F_k. Two different branch predicting schemes are used in the processor front-end respectively: one is the non-taken predictor commonly employed in embedded processors and the other is the perfect predictor used for comparison.

Fig. 3 shows the trend of the hit rate with the HTB size increasing from 32 instructions to 2048 instructions. The effects of increasing the HTB size are twofold: the live times of the instructions in HTB are extended, and the length of the captured trace is extended. Both of them contribute to the HTB hit count in different phases of program execution. Because of the increased instruction count from the mispredictions of the non-taken scheme, the hit rate of the non-taken predictor is substantially lower than that of the perfect predictor. A more sophisticated branch predictor will have a hit rate curve lies in-between the two curves presented in the figure. The hit rate result suggests that the HTB buffer can be used as a supplementing light-weighted storage for instructions since a large percent of the instructions can come from the HTB buffer.

Due to the reduced complexity and size, the HTB is far less power hungry than the instruction cache. If an instruction can be delivered from the HTB whenever a hit occurs, an energy-saving rate proportional to the hit rate can be achieved. Note that for the non-perfect predictors, the raw HTB hit rate also takes into account of the misfetched instructions. These instructions are useless for program execution but are inevitable for non-perfect predictors. However, delivering this type of instructions from the HTB instead of the instruction cache still brings energy savings.

In this paper, based on the above observations, we propose a novel scheme called *Trace Reuse (TR) cache* to improve the energy efficiency of instruction delivery for embedded processors. The TR cache is composed of a *history trace buffer* (HTB), which collects the instructions retrieved from the pipeline back-end of the processor, and a *trace entry table* (TET) for fast access to the trace buffer. The proposed TR cache structure resides at the same level of memory hierarchy as the conventional instruction cache, so no additional cache latency will be incurred. As a repeated trace execution is identified, the processor switches to the TR cache for low-power instruction delivery. Delivering the instructions in the form of traces, like the trace cache, enables the processor to sustain a higher instruction rate, and consequently improves the performance. With its simple structure, the TR cache represents a cost-effective design option, which is beneficial in both performance and energy efficiency for embedded processors.

The rest of this paper includes the following sections. Section II presents the design of the TR cache architecture and implementation issues. The performance and energy efficiency of the TR cache are evaluated in Section III. Related works and our contributions are discussed in Section IV. Finally, Section V concludes this paper.

II. Design of TR Cache

In this section, we present the TR cache architecture that is capable of delivering instructions from the HTB for embedded processors. Along with the HTB FIFO buffer which is used to store the retired instructions, a search mechanism is required to locate the reusable instructions in the FIFO buffer. A new instruction delivery process is also required to employ the TR cache as an additional instruction source. The TR cache architecture is presented in Section II-A and the detailed design options are discussed in Section II-B.

A. Architecture

The architecture of the TR cache is depicted in Fig. 4 where the conventional instruction cache is also illustrated to show the modification on the instruction delivery path. As mentioned in Section I (referring to Fig. 2), the stages following the IF stage are augmented with additional D flip-flop registers to bypass the undecoded instruction bits. In addition to these expanded stage registers required for buffering on-the-fly instructions, a simple multiplexer and mode switching logic are integrated into the fetch stage to select the proper instruction source. We present a TET design to index the HTB buffer for instruction reading. The TET is a small memory structure used to group the instructions in the HTB buffer so that they can be accessed in the granularity of traces. Specifically, the TET stores the trace-entry records each of which consists of the PC value of a control-transfer instruction and the corresponding HTB entry index. Note that the stored HTB index points to the instruction *following* the control-transfer instruction to mark a possible *trace entry*.

The contents of TET and HTB are updated as follows. Whenever an instruction is retired from the pipeline backend, it is buffered in the HTB along with its PC. If this newly retired instruction is a control-transfer instruction, a conditional branch for instance, a new trace-entry record will also be inserted into the TET as shown in Fig. 4. Since the HTB is managed in a FIFO fashion, the oldest instruction will be discarded to make room for a new one. If the discarded instruction is a marked trace entry, the corresponding (i.e., the oldest) record in TET will also be invalidated. The invalidation of an expired trace entry is crucial since otherwise the fetch logic will reference a trace entry which is no longer valid in the HTB. For the simplicity of explanation, at this moment we assume the TET has the enough capacity required to record all the trace entries in the HTB buffer. Design tradeoff for limiting the TET size will be presented in the next section.

By comparing the HTB in Fig. 4 to the superscalar model in Fig. 1, we can see that the HTB buffers the instruction stream in the same way that a reorder buffer of a superscalar processor does. This means that when the program executes a small loop, multiple instances of the same iteration are buffered in the HTB, which appears to be inefficient in using the memory. To improve instruction buffering efficiency, a more complex basic block threading mechanism such as the design in [21] may be used; however, such a design typically uses expensive associative lookup circuitry which raises both the power and area

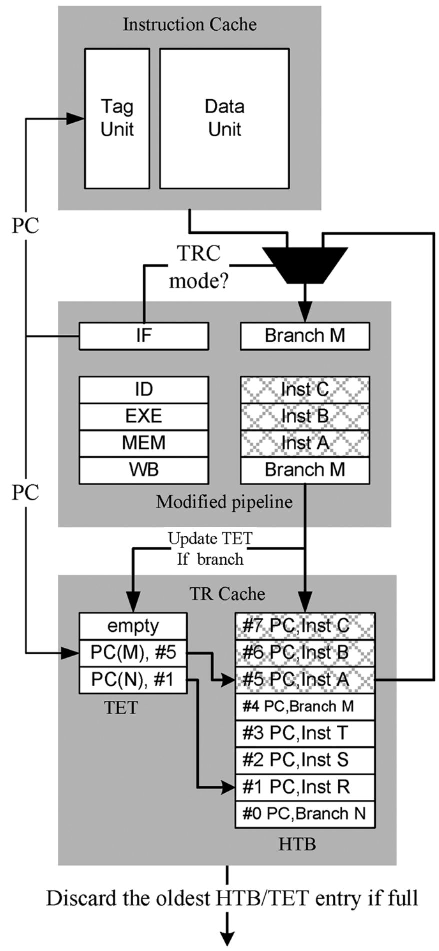

Fig. 4. TR cache architecture for embedded processor.

usage of the buffer. On the other hand, the HTB is a simple FIFO without the complex logic for associative lookups. The HTB uses a simple index-based access mechanism, to be presented later, which brings advantages in power and area usage as compared with an associative lookup-based design. The experimental results presented in Sections III-D and III-E show that the circuit simplicity of the TR cache offers significant power benefits.

To utilize the TR cache as an alternative source for instruction delivery, a new access mode is integrated to the fetch logic. Here we name the original access mode as the *cache mode* and the new one as the *TRC mode*. Fig. 5 shows the change in the flow of instruction delivery; the shaded blocks indicate new operations installed by the TRC mode. The processor is in cache mode by default, and for each cache-mode cycle the TET is searched in parallel with the cache (referring to Fig. 4). We assume the TET search can be completed in a single cycle. Our design option evaluations shown later validate this assumption. The validation of this assumption is presented in Section II-B.

The processor remains in cache mode until the TET search returns a hit. When a hit occurs, the HTB index returned by the TET will be latched and the processor will switch to the TRC mode at the next cycle. Note that for the cycle in which a TET hit event occurs, the control-transfer instruction is still delivered from the conventional instruction cache as illustrated in the upper part of Fig. 5. In the TRC mode, the fetch logic uses the

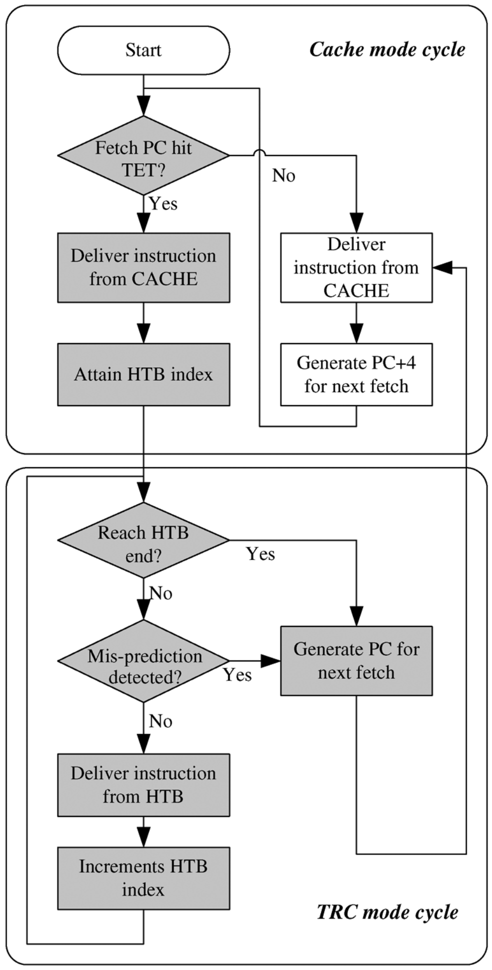
Start
Cache mode cycle
Fetch PC hit TET?
No
Yes
Deliver instruction from CACHE
Deliver instruction from CACHE
Attain HTB index
Generate PC+4 for next fetch
Reach HTB end?
Yes
No
Mis-prediction detected?
Yes
Generate PC for next fetch
No
Deliver instruction from HTB
Increments HTB index
TRC mode cycle

Fig. 5. Flowchart of instruction delivery.

HTB index instead of the PC value to get the instruction from the HTB buffer. For each TRC-mode cycle, the fetch logic performs two checks before actually acquiring the instruction: one is the availability of the instruction and the other is the validity of the current trace.

The availability of the incoming instruction is confirmed by checking the current HTB index against the HTB boundary pointers. If the index has reached the end of the HTB, the TRC-mode operation will be aborted and the next PC will be generated for the cache-mode operation. Moreover, the branch misprediction signal from the execution stage will also be checked to validate the ongoing trace. Provided that a branch misprediction is detected, the TRC-mode operation will be aborted and the target address forwarded from the execution stage will be used as the next PC. Note that for the branch misprediction recovery in the TRC mode, the processor is always reset to the cache mode and then fetches from the instruction cache. Though we can also initiate a new lookup into both the TET and HTB for additional chances of trace reuse, our experiment indicates that doing so only increases the overall HTB hit rate by about 2% and will incur additional recovery latency. Thus we decided to keep the recovery mechanism in TRC mode seamlessly integrated with the conventional pipeline operation.

TABLE I
MAXIMUM TET RECORD COUNTS FOR DIFFERENT HTB SIZES

HTB size (instructions)	32	64	128	256	512	1024	2048
Max. TET record count	15	22	41	72	135	261	507

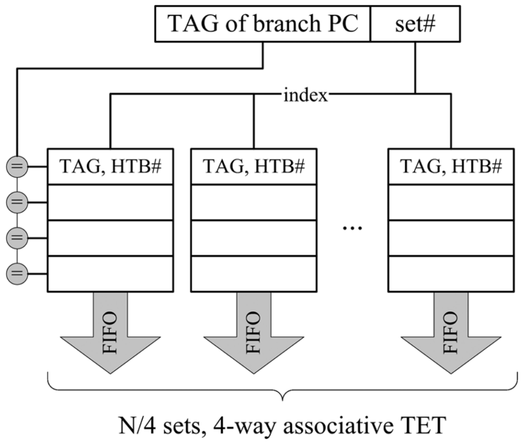

Fig. 6. Four-way TET with replaced-by invalidation policy.

B. Design Options of TET Implementation

As mentioned previously, the TET search is performed in each cache-mode cycle; therefore the size and the structure of the TET present two most important issues for the implementation of the TR cache. In the previous section, we assumed that the TET has enough capacity to accommodate records for every control-transfer instructions in the HTB. The TET size to be used is actually dependent on both the size of the HTB and the program behavior. Table I shows the maximum TET record numbers required for different HTB sizes, collected from the benchmark suite simulation described in Section III.

As shown in the table, the maximum record numbers converge to roughly one-fourth of the HTB size, thus we choose to limit the TET size to one-fourth of the HTB size. Apparently limiting the sizes of TET may cause the number of dynamic trace-entry records to exceed the size of the TET. For the TET, we propose a management scheme called *replaced-by-invalidation policy* to handle the case when the TET is full. That is, when the TET is full, instead of replacing any TET entry, the newly generated trace entry record is simply discarded until the oldest TET entry is invalidated by the removing of the expired trace-entry instruction in the HTB. This may cause some of the trace entry points not being recorded in the TET and therefore decrease the HTB hit rate. However, this policy will guarantee the invalidation of the expired TET entries which are then easily synchronized with the instruction retirement operation in the HTB.

For a small TET, the implementation of the TAG field can be of a traditional fully-associative structure, such as the low-power CAM design proposed in [22]. However, for a TET with a much larger capacity, e.g., a 512-entry TET with a 512×21 TAG array, this CAM structure will consume significant area

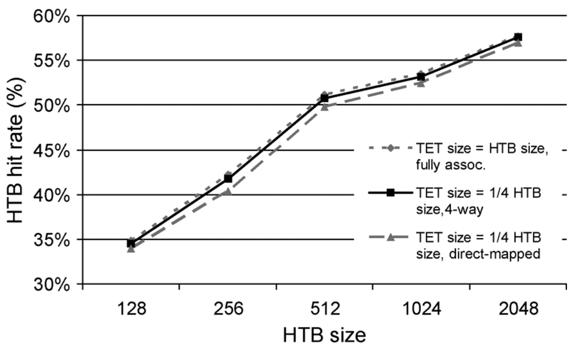

Fig. 7. HTB hit rates of different TET organizations.

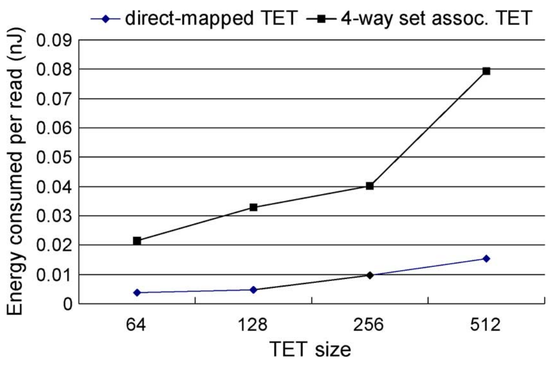

Fig. 8. Energy consumptions of different TET organizations.

and power usage not suitable for an embedded design. To reduce the overhead of implementing a TET, two SRAM-based alternatives including a four-way set associative design as shown in Fig. 6 and a direct-mapped organization (not illustrated) are evaluated. For the four-way set associative version, the TET of size N is physically divided into $N/4$ segments where the replaced-by-invalidation policy is enforced separately. When a specific set is full, a later record indexed to the same set is discarded. Likewise, the replaced-by-invalidation policy can also be applied to a direct-mapped design where each set contains only one entry. The effectiveness and energy consumption of these two simplified organizations are presented in Figs. 7 and 8, respectively, to show the design tradeoff. The results shown in Fig. 7 are collected from the simulation platform described in Section III; and the energy estimations in Fig. 8 are obtained using CACTI [25]. As shown in Fig. 7, the four-way organization TET is almost as effective as the fully-associative one. The direct-mapped version has caused an average of 2% drop in hit rate due to the conflict misses. However, the direct-mapped version provides over 80% of energy saving for each access as shown in Fig. 8. Obviously in this case, it would be preferable to trade energy saving with slight hit rate degradation.

Using the direct-mapped design can substantially reduce the circuit complexity and energy usage of the TET. However, the sequential order of the TET entries may be scattered due to the non-FIFO nature of the direct-mapped organization. This will hamper the in-order invalidation of the TET entries since the oldest entry may no longer be simply located at the tail of the buffer. We present a simple mechanism to solve this problem.

In Fig. 9(a), we show the content of a TET entry which includes a tag field for the PC of control-transfer instruction, a HTB number indexing the target instruction, and a *busy bit*

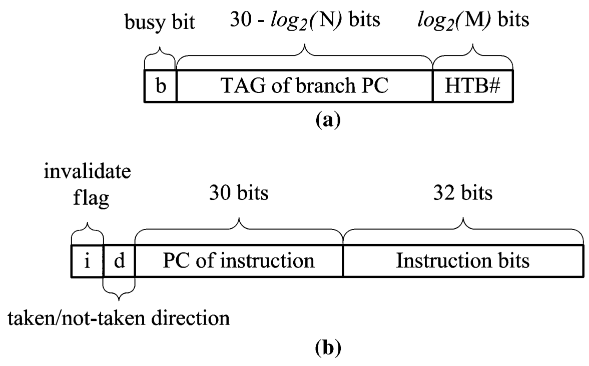

Fig. 9. Contents of TET and HTB. (a) Content of a TET entry; (b) content of an HTB entry.

which indicates whether the entry is occupied. Fig. 9(b) shows the content of a HTB entry which includes a PC-instruction pair, an invalidate flag and a taken/not-taken direction for the conditional branch instructions. Fig. 10 details the pseudo code for the replaced-by-invalidation policy. As a control-transfer instruction entering the HTB, its taken/not-taken history is logged as a reference for trace validity check. A TET index will be generated with the PC value of the control-transfer instruction. According to this index, the busy bit of the corresponding TET slot is checked to see if it is available for use. If the TET slot is available, the new trace-entry record for the control-transfer instruction will be inserted into TET and the invalidate flag of the corresponding entry in HTB will be set indicating that this instruction also occupies a TET slot. For control-transfer instructions which cannot get the available slot from the TET and all other instructions, their invalidate flags in the allocated HTB entries remain unset. After an instruction has traversed to the tail of the HTB and been chosen to discard, its invalidate flag will be checked. If the flag is set, the corresponding TET entry will be invalidated. The index number of the invalidated TET entry is generated using the PC of the instruction to be drained out of the HTB. This process will guarantee an in-order invalidation of the TET contents which is then synchronized with the instruction retirement in the HTB.

III. Experimental Results

In this section, we present the experimental methodology and the results. To demonstrate the feasibility of the pipeline modification described in Section II-A, the additional D flip-flops in each pipeline stage and the multiplexing logic in the instruction fetch stage are implemented in an ARM-compatible five-stage RISC core described in [23]. The new RTL code of the modified processor core, which includes the register file and the complete pipelined datapath, is then synthesized using the Synopsys Design Compiler along with the UMC 90-nm technology library. Note that the caches and the TLBs are excluded from the synthesis process to assess the area overheads on the processor integer core. Table II lists the synthesis results of the modified processor core and the comparisons with the original design. With the nonpipelined multiplier and adder provided by the Synopsys DesignWare, the synthesized core is able to operate at a clock period of 6 ns (166 MHz). The cell area of the modified pipeline increases only about 5.4% of the original pipeline. The

```
function HTB insertion (instruction PC, instruction bits)
BEGIN
  read the HTB FIFO head pointer h;
  update the PC field of the h_th HTB entry;
  update the instruction bits field of the h_th HTB entry;
  advance the HTB FIFO head pointer h;

  if (instruction is control-transfer type)
      record the taken/not-taken direction d
        generate index i from the instruction PC;
        generate tag t from the instruction PC;

        if (the busy bit in the i_th TET entry is 0)
            update the i_th TET entry with t and h;
            set the busy bit in the i_th TET entry to 1;
            set the invalidate flag field in the h_th HTB entry to 1;
      else
            set the invalidate flag field in the h_th HTB entry to 0;
            return;

     else
       set the invalidate flag field in the h_th HTB entry to 0;
       return;
END
```

```
function HTB retirement
BEGIN
  read the HTB FIFO tail pointer t;

  if (the invalidate flag of the t_th HTB entry is 1)
     generate index i from the PC field of the t_th HTB entry;
     set the busy bit in the i_th TET entry to 0;

  advance the HTB FIFO tail pointer t;
  return;
END
```

Fig. 10. Pseudo code for the replace-by-invalidation policy.

TABLE II
SYNTHESIS RESULTS OF THE MODIFIED RISC PIPELINE

	Original (166 MHz)	After mod. (ratio%)
Cell area	203323	214405 (105.4%)
Estimated dynamic power	3.64 mW	4.24 mW (116.4%)
Estimated leakage power	581 uW	612 uW (105.3%)
fetch stage critical path latency	0.38 ns	0.38 ns (100%)

estimated dynamic power increases about 0.6 mW, which is insignificant in comparison with the 10.2 mW power consumption of a direct-mapped 16 kB instruction cache running at 166 MHz. The synthesis tool has reported different critical paths in the fetch stage for the two versions of the core, with unchanged latency. This shows that the additional multiplexing logic does not pose noticeable timing issue to the instruction fetch logic for this implementation.

Moreover, our synthesis results indicate that, using the UMC 90-nm library, the latency of the multiplexing logic itself is 0.26 ns. For high-speed designs such as an operating frequency of 1 GHz, for instance, the fetch logic and access to the instruction cache itself will be pipelined and the reported MUX latency still leaves quite an adequate margin for the integration of other logic such as the address decoder of the cache.

To further reveal the impact on performances, the TR cache architecture is integrated into the Simplescalar-ARM [24] platform for the simulation of a detailed cycle-accurate model. The baseline processor is configured using the parameters listed in Table III to model an embedded processor similar to the StrongARM [14]. A set of benchmark programs selected from MiBench [27] listed in Table IV is used to evaluate the IPC and energy consumption of the TR cache. The static code sizes and the used input sets of each program are also detailed in the table. All programs are run to completion.

TABLE III
BASELINE PROCESSOR CONFIGURATION

Issue width	1 (in-order)
Branch predictor	Non-taken
Mis-prediction latency	3 cycles
Integer ALU / multiplier	1 / 1
Floating-point ALU / multiplier	1 / 1
Level-1 Cache	Harvard architecture, 1 cycle hit 16KB for instruction[a], 16KB for data 32-byte cache line, 32-way, FIFO replacement
Level-2 cache	none
Memory bus width	32 bits
Memory latency	64 cycles for first chunk, 1 cycle burst

[a]Reduced instruction cache sizes including 8, 4, and 2 kB are also employed for the scalability evaluations.

The energy consumptions, areas, and access times of the suggested memory structures described in Section II are estimated using CACTI v5.3 [25]. The CACTI is an integrated memory modeling tool, which can provide optimized circuit layout parameters and resultant physical characteristic such as access time, dynamic access energy, and leakage power by user-defined specifications. Table V shows the general parameters used in CACTI and the explanations for these options can be found in [26]; organization parameters which vary from module to module such as total capacity and associativity are not shown in the table. The derived estimations are used for the evaluations of performance and energy consumption of the TR cache and other referenced models.

A. Access Time and Area Estimations

Using CACTI, both the TET and HTB are modeled as SRAM modules with one exclusive read port and one exclusive write port. The areas, access times, and power usages of other referenced memory modules are also listed in Table VI. The 16 kB/32-way instruction cache is modeled according to the baseline organization listed in Table III, while the 8, 4, and 2 kB ones correspond to the reduced instruction cache modules depicted in Fig. 13.

It can be observed that the access times of the direct-mapped TETs are far less than that of the 16 kB/32-way instruction cache. This shows that the TET search in parallel with the cache mode operation has no impact on the processor cycle time. The access time of the largest HTB (2048 entries simulated) also lies in a decent safe margin, guaranteeing that the processor cycle time in the TRC mode will not increase either. The 16 kB/32-way instruction cache has an access time of 2.148 ns, which lies in the safe margin of operating at a frequency of 450 MHz. Therefore we assumed the operation frequency of the processor to be 450 MHz and used this assumption in the calculation of leakage power in Section III-D.

TABLE IV
SELECTED MIBENCH PROGRAMS

Category	Selected programs	Code size (KB)	Input set
Automotive (5 programs)	bitcount	254	1125000
	quicksort	246	qsort_large
	susan-smooth, susan-edge, susan-corner	233	input_large.pgm
Consumer (8 programs)	jpeg-c	291	input_large.ppm
	jpeg-d	303	input_large.jpg
	lame	1620	large.wav
	madplay	291	large.mp3
	tiff2bw	271	large.tif
	tiff2rgba	297	large.tif
	tiffmedian	286	large.tif
	tiffdither	282	largebw.tif
Network (2 programs)	dijkstra	234	dijkstra_large
	patricia	238	large.udp
Office (4 programs)	ghostscript	1061	large.ps
	ispell	235	large.txt
	say	278	largeinput.txt
	search	184	search_large
Security (4 programs)	blowfish-e, blowfish-d	186	large.asc, large.enc
	rijndael-e, rijndael-d	194	input_large.asc, input_large.enc
Telecomm (8 programs)	sha	183	input_large.asc
	adpcm-c, adpcm-d	182	large.pcm, large.adpcm
	crc	182	large.pcm
	fft, fft-inv	189	8, 32768
	gsm-toast, gsm-untoast	235	large.au, large.gsm

TABLE V
GENERAL PARAMETERS APPLIES IN CACTI

Technology	90 nm
Operation temperature	310°K
Access mode	Normal
Transistor type	ITRS-HP[28]
Interconnect projection type	Conservative
Type of wire outside mat	Semi-global

The area estimations are also presented in Table VI and are normalized to that of the 16 kB/32-way instruction cache in the baseline processor for comparison. As shown in the table, the 2048-entry HTB contains as much data capacity as the 16 kB/32-way instruction cache but possesses only about one-third of the area. At the same level of data capacity, the TR caches possess substantially less areas than the instruction caches do due to the low-complexity circuitry. This means it could bring options to relax the area constraint of the processor if the TR cache can be used as a replacement or partial replacement of the instruction cache. The feasibility of replacing a portion of the instruction cache with a TR cache will be discussed in later section.

B. IPC Performance Analysis

The average IPC of the 31 programs listed in Table IV is used as the performance metric. Due to the large variations in the elapsed cycle counts of the programs, we use (1) instead of arithmetic mean to calculate the average IPC

TABLE VI
ESTIMATED ACCESS TIMES AND AREAS OF THE REFERRED MEMORY MODULES

TET size / capacity	Access time (ns)	Area (mm^2)	Area ratio (%)
512 entries / 2560 bytes	0.7648	0.11641	6.7%
256 entries / 1280 bytes	0.6673	0.08092	4.6%
128 entries / 640 bytes	0.7218	0.03357	1.9%
64 entries / 320 bytes	0.6219	0.02263	1.3%
HTB size / capacity	Access time (ns)	Area (mm^2)	Area ratio (%)
2048 entries / 16384 bytes	0.9267	0.59958	34.5%
1024 entries / 8192 bytes	0.8396	0.32015	18.4%
512 entries / 4096 bytes	0.8006	0.17702	10.1%
256 entries / 2048 bytes	0.7559	0.11513	6.6%
128 entries / 1024 bytes	0.5842	0.08432	4.8%
Instruction Cache organization / capacity	Access time (ns)	Area (mm^2)	Area ratio (%)
16KB, 32-way / 16384 bytes	2.1488	1.73692	100%
8KB[a], 16-way / 8192 bytes	1.2818	0.57546	33.1%
4KB[a], 8-way / 4096 bytes	0.9103	0.28894	16.6%
2KB[a], 4-way / 2048 bytes	0.7285	0.15395	8.8%
16KB[b], direct-mapped / 16384 bytes	0.8470	0.59168	34.1%
BTB and Filter Cache organization / capacity	Access time (ns)	Area (mm^2)	Area ratio (%)
256-set, 4-way/ 4096 bytes (branch target buffer[b])	1.3096	0.22660	13.7%
512-set, direct-mapped / 2048 bytes (BTB)	0.7937	0.15735	9%
16-set, direct-mapped / 512 bytes (filter cache[b])	0.6706	0.05020	2.9%

[a]discussed in Section III-B; [b]discussed in Section III-D.

$$\mathrm{IPC}_{\mathrm{average}} = \frac{\sum I_k}{\sum C_k} \tag{1}$$

where the I_k is the number of valid dynamic instructions of the K_{th} program; and the C_k is the elapsed cycle count during the execution of the K_{th} program. The IPC in (1) is equivalent to the summation of the weighted IPC of each program.

To also compare the performance of the TRC-based models to the high-end embedded processors such as ARM's Cortex R series [29], two branch-predictor-based configurations, namely bimodal-1K and gshare-4K, are also evaluated. As reported in [1], the bimodal predictor is a basic dynamic prediction scheme which saturates at around 93.5% and the gshare is a refined global predictor which can achieve prediction accuracy up to 96.5% with larger table size. For the 31 benchmark programs evaluated in our work, the bimodal predictor saturates at the prediction rate of 93.38% with 1k table size while the gshare predictor achieves 94.17% with 4k table size. Fig. 11 shows the average IPC of the baseline processor and the proposed TR cache architecture. The label $\mathrm{TRC} - \mathrm{M}$ designates a TR cache with an HTB size of M instructions and a TET size of $M/4$ trace-entry records. The IPC results of using realistic branch predictors, which are labeled as bimodal-1K and gshare-4K respectively, are also shown in the figure. The bimodal-1K system models a baseline processor employing a 1024-entry bimodal predictor and a 1024-entry branch target buffer, while the gshare-4K models the 4096-entry gshare predictor table with a 512-entry branch target buffer.

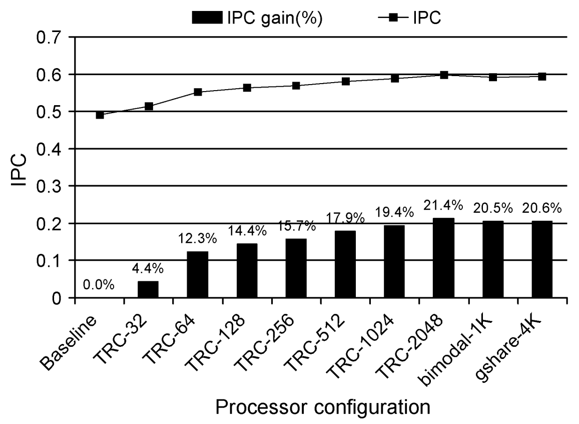

Fig. 11. Average IPC and improvement rate.

As shown in the figure, incorporating the TR cache provides significant performance improvement over the baseline processor. The improvement, which is proportional to the size of the TR cache, mainly comes from the locality of the program and the inherent branch prediction effect introduced by reusing the history trace. For instance, the TRC-1024 model has performed as well as the two predictor-based models.

We define a metric called TRC effective instruction rate to indicate how many of the executed instructions truly come from the TR cache. For all the instructions fetched, only the valid ones are able to reach the pipeline backend. In (1), the number of the valid instructions during the execution of the K_{th} program is denoted as I_k. Among the I_k valid instructions, there are T_k instructions which are delivered from the TR cache. The *TRC effective instruction rate* is then calculated as follows:

$$\text{TRC effective instruction rate} = \frac{\sum T_k}{\sum I_k} \times 100\%. \qquad (2)$$

Specifically, the TRC effective instruction rate defines the ratio of the useful instructions delivered from the HTB over the total number of instructions that can reach the write-back stage (that is, only the valid instructions excluding the mispredicted ones). So it can measure the effectiveness of a TRC mechanism, for instance, when different TET schemes or different HTB sizes are used. In other words, the TRC effective instruction rate reflects how well the buffered history traces in the TR cache can be utilized.

As we know that the basic idea behind branch prediction is to extract useful branch targets from the history trace. In our work the same effect is produced when a valid branch target is delivered from the TR cache. To reflect the exhibited branch prediction performance, we define the equivalent branch prediction rate of a TR cache as follows: the number of the correctly-predicted ones divided by the total number of control-transfer instructions. Fig. 12 shows the TRC effective instruction rate and the achieved equivalent branch prediction rate for various TR cache configurations. First, we note that even a small TR cache such as TRC-64 is able to provide about 45% of the effective instruction rate and about 65% in the equivalent branch prediction rate. As a result, this improves the IPC about 12% compared to the baseline one.

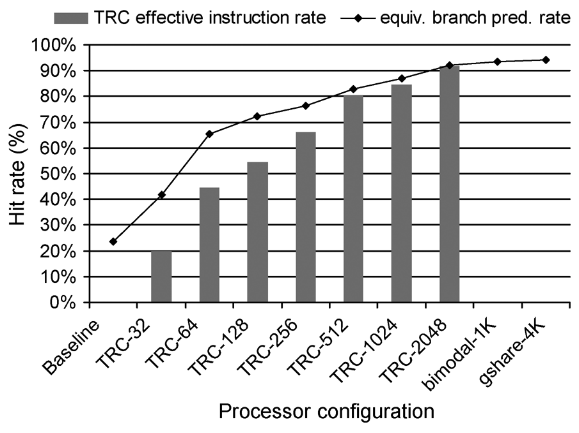

Fig. 12. TRC effective instruction rates and equivalent branch prediction hit rates.

We further observed that over 80% of the valid instructions can be delivered from the TR cache when the TRC size is larger than 512 instructions. With this, we inferred that the TR caches have the potential to substitute the conventional instruction cache and even have the capability to tolerate the performance loss when using a smaller instruction cache. To further explore this hypothesis, we also present the IPC results of the processor with reduced instruction caches of 8, 4, and 2 kB, respectively.

Fig. 13 depicts the performance impact of reducing the instruction cache size for various TR cache models. The IPCs of the TR cache models are normalized to the baseline processor which only employs non-taken branch prediction. The instruction cache size is reduced by halving the associativity (i.e., the number of ways) to demonstrate the effects of both capacity misses and conflict misses. The left-most column shows the normalized IPCs of 16 kB/32-way cache, and the remaining columns depict those of the 8 kB/16-way, 4 kB/8-way and 2 kB/4-way, respectively. Note that for the predictor-based models, only the curve of the gshare-4 K model is shown in the figure since the two models perform very closely (within 0.2% difference) for all the elaborated cache sizes and the gshare-4K achieves just slightly a better performance.

It can be observed that the performance of the processor with a non-taken predictor drops rapidly as the instruction cache is reduced while the TRC-based models and the branch-predictor-based model exhibit different capability to compensate the IPC drop. Among them the TRC-2048 model exhibits the best performance in sustaining the IPC when the instruction cache size is reduced. On the other hand, the performance of the predictor-based model depends strongly on the size of the instruction cache. For the cases of 16 and 8 kB instruction caches, the gshare-4K performs almost as well as TRC-2048; however, when the cache is reduced to 4 kB, its IPC drop to that of TRC-1024; and finally when the cache is reduced to 2 kB, even the TRC-512 can outperform the gshare-4K model.

With similar accuracy of branch prediction, the advantage of TR cache over branch-predictor-based model is more evident when the instruction cache capacity is insufficient to contain the application kernel. For example, the TRC-2048 model outperforms the gshare-4K by 25% and 50% for the cases of 4 and 2 kB instruction cache, respectively. This can be explained with the

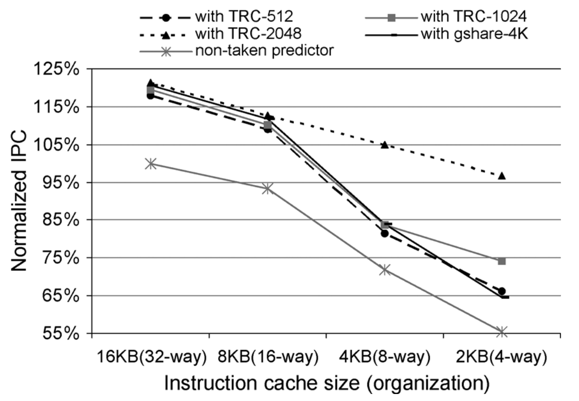

Fig. 13. Performance impact of reducing instruction cache size.

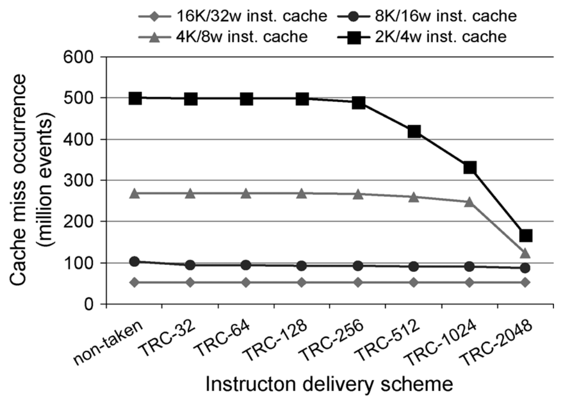

Fig. 14. Total cache miss events of different instruction cache sizes.

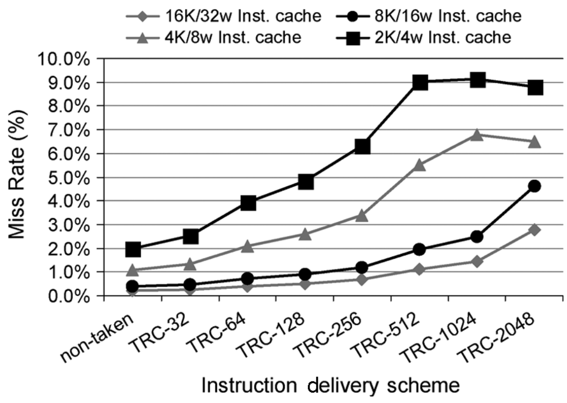

Fig. 15. Miss rates of different instruction cache sizes.

TABLE VII
INSTRUCTION CACHE ACCESS COUNTS OF DIFFERENT DELIVERY SCHEMES

Delivery scheme	Total access (million events)	Ratio (%)
Non-taken	25553.59	100.00%
TRC-32	19956.24	78.10%
TRC-64	12793.33	50.06%
TRC-128	10328.04	40.42%
TRC-256	7835.099	30.66%
TRC-512	4685.91	18.34%
TRC-1024	3656.46	14.31%
TRC-2048	1901.86	7.44%

fact that the HTB virtually expands the capacity of the conventional instruction cache in a dynamic manner. Through a simple yet effective access scheme provided by TET, such a dynamically expanded cache space can deliver even more performance gain than a dynamic branch prediction scheme.

C. Impact on Instruction Cache Access

Since the TRC mechanism is used as an alternative instruction source, naturally it affects the instruction cache access statistics. Table VII lists the total number of instruction cache accesses for the 31 programs simulated of different TRC sizes. The access counts to instruction cache are substantially reduced as the capacity of the TR cache increases, and for the best case, up to 92.5% of the accesses can be eliminated by TRC-2048. As more targeted accesses being shifted from the instruction cache to the TR cache, the energy saving effect of the TRC becomes more evident. A detailed energy efficiency analysis is given in the next section.

In the previous section, we mentioned that the TRC-512, TRC-1024, and TRC-2048 deliver performance advantages when the instruction cache capacity is too small to store the application kernel. This observation can be further supported by looking into the cache miss numbers. Fig. 14 shows the total cache misses of the 31 programs simulated for the four different sizes of instruction cache configurations. For the cases of 16 and 8 kB instruction caches, the major parts of the application kernels can be well contained in the instruction cache so that the cache miss number remains approximately constant across all delivery schemes. However, for the 4 and 2 kB cases, the cache miss occurrences of the non-taken model increase exponentially due to insufficient capacity and address conflicts. Under such circumstances, the TRC-2048 is able to eliminate more than 50% of the cache miss events.

Fig. 15 depicts the change of cache miss rates as the TRC capacity increases. For 16 and 8 kB caches, since the miss occurrences remain approximately constant (referring to Fig. 14) and the total number of instruction cache accesses is substantially reduced (referring to Table VII), the miss rates of the instruction cache are raised accordingly. However, for the cases of 4 and 2 kB cache, the miss rates regress at TRC-2048. This indicates that the cache miss elimination effect of TRC-2048 slightly surmounts the effect of cache access reduction.

D. Energy Efficiency

In this section, we present the evaluation of energy consumption and normalized energy-delay product of using the TR cache and other reference configurations. The energy consumption of the fetch logic mainly comes from the power dissipation of the instruction cache and the augmented memory structures such as the TET and HTB. We have developed a power model of the fetch logic from the energy usage listed in the following to calculate the total fetch energy:

$$\begin{aligned} E_{\text{dynamic}} &= N_{\text{read}} \times E_{\text{read-per-port}} \\ &\quad + N_{\text{write}} \times E_{\text{write-per-port}} \qquad (3) \\ E_{\text{static}} &= P_{\text{leakage}} \times T_{\text{program-execution}} \qquad (4) \\ E_{\text{module}} &= E_{\text{dynamic}} + E_{\text{static}} \qquad (5) \\ E_{\text{total}} &= \sum E_{\text{module}}. \qquad (6) \end{aligned}$$

The $E_{\text{read-per-port}}$, $E_{\text{write-per-port}}$, and P_{leakage} in the above equations stand for read energy per port, write energy per port and leakage power respectively, and are estimated by the CACTI tool [25]. The parameters used in estimating these values are described in Section III-A. The N_{read} and N_{write} stand for the number of read accesses and the number of write accesses of the evaluated memory structures respectively. These two numbers represent a model of the dynamic activity factor which helps to reflect the impact on energy consumption caused by different access behavior. $T_{\text{program-execution}}$ is the elapsed program execution time which is used to estimate the change in leakage energy due to performance improvement. These program-related parameters are obtained through the simulations using the Simplescalar platform described in previous section.

For each referred memory module, the total energy used, E_{module} is derived by summing up the dynamic access energy E_{dynamic} and the static leakage energy E_{static}; and then in (6) all the individual E_{module} are summed to present an overall energy usage in the fetch logic during the program execution. Finally, the energy-delay product (EDP) is calculated by multiplying the normalized E_{total} and the normalized $T_{\text{program-execution}}$ to reflect the overall energy efficiency of each configuration. For the EDP, the smaller the better of a configuration in energy efficiency.

As mentioned earlier, the TR caches provide energy savings in the same way a filter cache does; therefore the filter cache design is also evaluated for comparison. The original filter cache proposed in [16] suffers from IPC drop due to the increased cache latency. Various schemes, including software-based [17], [18], [20] and hardware-based ones [19], [30], are proposed to mitigate the IPC degradation of filter cache. Since the TR cache is simply a hardware-oriented scheme, for our comparisons we select a recently proposed filter cache design that also uses a hardware-based improvement scheme [30]. The organization and utilization strategy proposed in [30] provide the effectiveness similar to a dynamic loop cache [18] with the design complexity of a direct-mapped cache, thus represents a contemporary filter cache design.

Fig. 16 presents the fetch energy usage rates, which are the normalized forms of the values calculated with (3)–(5), of a 16 kB 32-way instruction cache when different instruction delivery schemes are applied. This 16 kB 32-way instruction cache corresponds to the one employed in the baseline processor listed in Table III. As described in [30], the filter cache size is set to 1/W of the cache size where W is the number of ways, hence the filter cache size for the 16 kB/16-way cache evaluated in [30] is 1 kB and an energy reduction rate of 67% is reported. Since the baseline model in our work employs a 16 kB/32-way instruction cache, based on [30] a direct-mapped filter cache with 512-byte capacity is applied for our experiment.

In Fig. 16, it can be seen that the total energy usage of the instruction cache is substantially reduced by the TR cache and the effect grows with the increase of the TR cache size. For the best TR cache configuration, the TRC-2048, about 75% of energy saving is achieved while the filter cache achieved 80% saving. Taking the reduced delay time into account, the TRC-2048 has achieved the energy-delay product value of 0.207, which is very

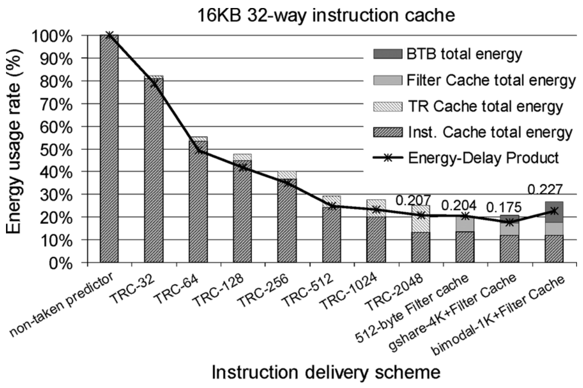

Fig. 16. Fetch energy usage rates and EDPs of suggested instruction delivery schemes with 16 kB 32-way instruction cache.

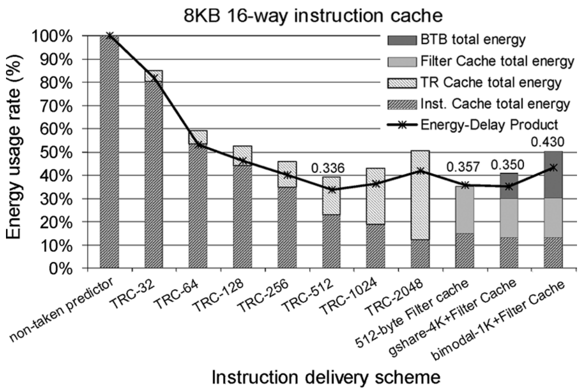

Fig. 17. Fetch energy usage rates and EDPs of suggested instruction delivery schemes with 8 kB 16-way instruction cache.

close to the value of 0.204 achieved by the filter cache. This result indicates that in addition to the IPC improvement, the TR cache is able to achieve energy efficiency close to the contemporary filter cache design as well. Moreover, the energy usage of two hybrid configurations, which incorporates both the branch-predictor models mentioned in Section III-B and the filter cache, are also presented. Note that the gshare-4K uses a 512-entry BTB, which has a lower complexity and capacity than that of the bimodal-1K model (referring to Fig. 11), yet it achieves the same amount of execution time reduction, therefore outperforms the bimodal-1K and all other models for both energy saving and EDP.

To further explore the effectiveness of the TR cache and the other reference configurations, we also provide the energy usages of the fetch logic when the instruction cache is scaled down in both size and complexity. Figs. 17–19 show the fetch energy usage rates when the instruction cache is reduced to 8 kB/16-way, 4 kB/8-way, and 2 kB/4-way, respectively. These reduced instruction caches correspond exactly to those described in Section III-B (referring to Fig. 13), in which their numbers of ways are halved in turn while the set number of a single way remains the same. This setting also keeps the size of the applied filter cache constant, in our case, 512 bytes.

In Table VIII, we can see that the total fetch energy dissipated in the instruction cache of the processor with non-taken predictor drops dramatically when the size of the instruction cache is reduced. This is why the energy usage rates of the TR caches become more dominant in Fig. 17 and Fig. 18. This is

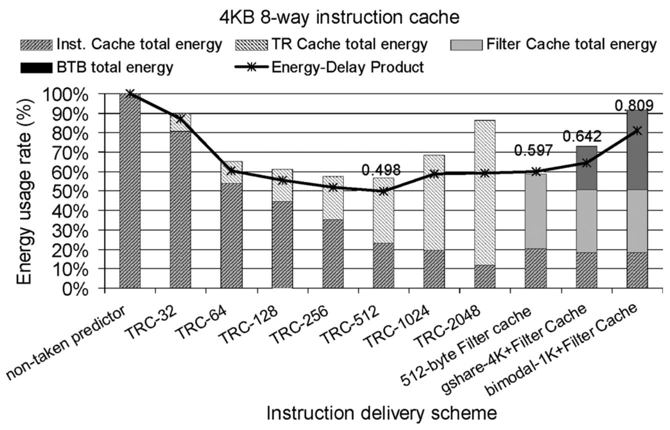

Fig. 18. Fetch energy usage rates and EDPs of suggested instruction delivery schemes with 4 kB eight-way instruction cache.

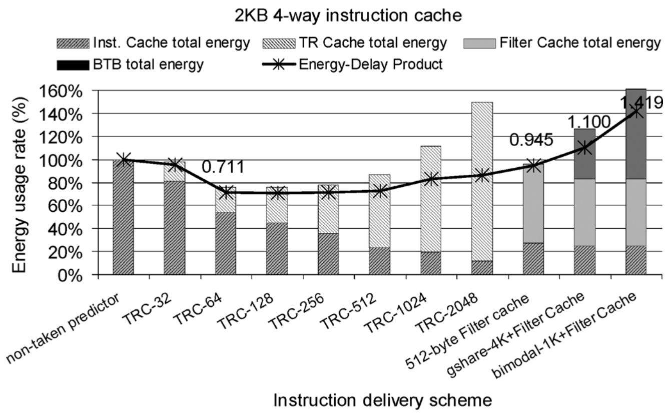

Fig. 19. Fetch energy usage rates and EDPs of suggested instruction delivery schemes with 2 kB four-way instruction cache.

TABLE VIII
E_{total} OF THE PROCESSOR WITH NON-TAKEN SCHEME

Instruction Cache size / organization	E_{total} (Joul)
16KB / 32-way (Baseline)	10.873
8KB / 16-way	3.468
4KB / 8-way	1.802
2KB / 4-way	0.991
16KB / direct-mapped	2.012

also true for the branch-predictor based models since the energy consumed by the branch target buffer becomes more significant. In Fig. 19, the fetch energy consumed by TRC-1024, TRC-2048, and the two branch-predictor based models even exceeded that of the non-taken configuration.

Taking the energy-delay product as a criterion, the gshare-4 K model performs better than the TRC-2048 with the support of an additional filter cache, for the cases of 16 kB/32-way and 8 kB/16-way instruction cache. However, when the size of the instruction cache is further reduced, in our case, 4 kB/8-way and 2 kB/4-way, the TRC-2048 is able to save more execution time and thus achieve better EDP. This shows that under the circumstance of reduced instruction cache size, the TRC-based solution will deliver better energy-efficiency than the combination of branch predictor and filter cache.

It can also be observed that the TR cache presents good adaptability when the HTB size is properly selected under different sizes of instruction cache. Specifically, the TR cache

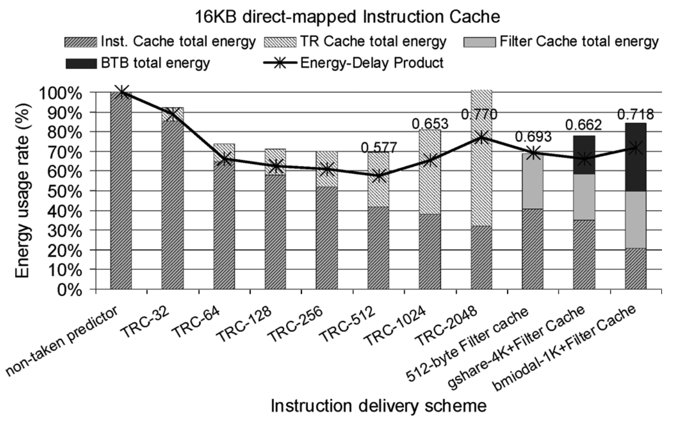

Fig. 20. Fetch energy usage rates and EDPs of suggested instruction delivery schemes with 16 kB direct-mapped instruction cache.

configuration with the best energy-saving rate in Figs. 17 and 18 are TRC-512 while in Fig. 19 the TRC-64 performs best. The TRC-512 can save up to 60% and 43% energy for the 8 kB/16-way and 4 kB/8-way instruction cache respectively. For the 2 kB/4-way instruction cache, the TRC-64 is able to achieve 25% energy saving while the filter cache can only achieve 5%. It can be observed that with proper selection of size, the TR cache is able to achieve better energy-saving effect than the filter cache. Although in the cases of 2 kB instruction cache, the TRC-1024 and TRC-2048 consumes excessive energy, the resultant energy-delay product values still match up to that of the filter cache. This indicates that the reduced delay times introduced by the TR cache are more than enough to compensate the exceeded energy consumptions.

The StrongARM processor model which is used as the baseline in our experiments employs a relatively complex 32-way instruction cache to compensate the performance disadvantage of its simple pipeline. Therefore the proposed TR cache is able to reduce the access energy due to its simple FIFO structure. As shown in the figures and analysis above, this effect will gradually diminish if the complexity and capacity of the employed instruction cache is lowered. To investigate the power-saving ability of TR cache when cooperated with an instruction cache with the same capacity as the baseline but much less complexity, the energy usage rate of a direct-mapped 16 kB instruction cache with various instruction delivery schemes are presented in Fig. 20. The TRC-512 outperforms all other configurations by achieving 31% energy reduction and the lowest EDP of 0.577.

E. Cost Analysis and Discussions

In this section, we present the area cost analysis of the various combinations of instruction cache organizations and instruction delivery schemes presented in the previous sections. The combinations selected for our comparison are those that achieve preferable results of either IPC gain, EDP, or both. Each of the combinations is considered a possible alternative design option. We divide these design options into two groups to emphasize two different preferences in design strategy: the performance-oriented group focuses on the IPC gain while the energy-efficiency group seeks to trade IPC with better energy-delay product.

TABLE IX
PERFORMANCE-ORIENTED DESIGN OPTIONS

Design Option	Inst. Cache capacity / Instruction delivery scheme	Area cost (%)	IPC gain (%)	EDP [a]
P1	16KB 32-way/ TRC-2048	141.2%	21.4%	0.2069
P2	16KB 32-way / gshare-4K	109.1%	20.6%	0.6918
P3	16KB direct-mapped / TRC-2048	75.3%	19.7%	0.1297
P4	16KB 32-way / gshare-4K+Filter cache	111.9%	17.7%	0.1754
P5	16KB direct-mapped / TRC-512	46.2%	10.2%	0.0974
P6	16KB direct-mapped / gshare-4K+Filter cache	46.1%	7.4%	0.1175
Baseline	16KB 32-way / non-taken	100.0%	0.0%	1.0000
FC	16KB 32-way / Filter cache	102.9%	-1.8%	0.2038
DM	16KB direct-mapped / non-taken	34.1%	-8.7%	0.2026

[a] All data including EDPs given in this table are normalized to the baseline model

TABLE X
ENERGY-EFFICIENCY DESIGN OPTIONS

Design Option	Inst. Cache capacity / Instruction delivery scheme	Area cost (%)	IPC gain (%)	EDP [a]
Baseline	16KB 32-way / non-taken	601.1%	39.4%	4.3296
REF	4KB 8-way / non-taken	100.0%	0.0%	1.0000
E1	2KB 4-way / non-taken	53.3%	-22.6%	0.7104
E2	2KB 4-way / Filter cache	70.7%	-24.0%	0.6929
E3	4KB 8-way / TRC-64	121.2%	8.6%	0.6013
E4	4KB 8-way / Filter cache	117.4%	-1.4%	0.5972
E5	2KB 4-way / TRC-64	74.5%	-17.6%	0.5053

[a] All data including EDPs given in this table are normalized to the REF model

The performance-oriented design options are listed in Table IX, which is sorted by the IPC gain column. The areas of the modules referred in each option are summed up and normalized to the baseline model to present the overall cost. Besides, the energy-delay products are also collated for the comparison of energy efficiency. For the aggressive design strategy of achieving better throughput and energy efficiency at the expense of additional area cost, the TRC-2048 (option P1) delivers the best IPC gain and an impressive EDP improvement.

If we consider a more conservative design strategy, for which not to increase overall area is preferred, downgrading the instruction cache to trade the area budget for the TR cache is also beneficial. With the TR cache, the performance loss due to downgraded instruction cache can be well compensated. As shown in the table, downgrading the instruction cache from 32-way to direct-mapped results in 8.7% IPC loss (referring to option DM); however by integrating a 2048-entry TR cache with the direct-mapped cache, option P3 is able to sustain the IPC at the same level of the branch-predictor based models (option P2 and P4), along with less area cost and better EDP. For a more stringent area budget, option P5 outperforms P6 in both performance and energy efficiency. It can be observed that the advantage of the TR cache over the hybrid model of branch predictor and filter cache grows as the instruction cache is downgraded from 32-way to direct-mapped.

Table X shows the design options for the energy-efficiency strategy and the table contents are sorted by the energy-delay product column. Compared to the baseline model, option REF has much lower circuit complexity and area cost which are commonly found in designs with low-power profile and therefore is considered the basic reference of this table. The other reference option E1 is the processor with a non-taken predictor and instruction cache size of 2 kB which represents the lower bound of area cost discussed in this paper. By comparing option E1 and option E2, it can be observed that the filter cache provides very little EDP improvement for the 2 kB instruction cache. With roughly the same area cost, the option E5 which incorporates TRC-64 is able to provide better energy efficiency and performance than E2. For those willing to exchange more area cost for less IPC loss, the combination of a 4 kB instruction cache and TRC-64 in option E3 will provide better performance, and energy efficiency very close to the filter cache does in option E4. Obviously the TR cache is also an attractive design option for the energy-efficient design strategy.

IV. RELATED WORK

The previous publications related to our work can be classified into four categories: filter cache, trace cache, instruction reuse, and branch resolution. In this section, a brief survey will be presented, and discussions of the differences between these techniques and the proposed TR cache will be given.

A. Filter Cache

In 1997, Kin proposed the filter cache [15], [16] and indicated the concept of trading performance with power efficiency. The original design of the filter cache uses a small level-zero cache, which dissipates less power than the level-one cache, to filter out the majority of the cache accesses. The accesses which hit in the filter cache will save significant energy, but with the price of increased miss latency. Bellas later proposed the loop cache [17], to filter out the loop code precisely with compiler support. The loop cache resides at the same level as the conventional instruction cache; however the processor depends on hints provided by the compiler to load and access the loop cache. A prediction scheme used to decide whether or not to access the filter cache is also proposed to mitigate the performance degradation [19]. Tang further refined the predictive filter cache into a decode filter cache [20], which stores the decode instruction to provide more aggressive energy saving effect. Recently, Janapsatya proposed the HitME [30] buffer structure, which avoided the performance degradation via a novel replacement policy.

The TR cache provides energy reduction effect in the same way these filter cache based schemes do. But either the filter cache or the loop cache depends on filtering the frequently accessed instructions from the level-one cache; by contrast the TR cache directly retrieves the instructions from the back-end of the processor. Furthermore, the filter cache is accessed in the granularity of instruction while the TR cache is indexed with traces. Utilizing the trace as the minimum access unit sacrifices hit rate but greatly simplifies the management hardware. Moreover, the trace-based management enables a simple yet effective branch prediction capability and provides significant performance gain.

B. Trace Cache

Rotenberg *et al.* proposed the trace cache [10] to break the bottleneck of fetch bandwidth in superscalar processors. The concept of rearranging the instruction cache contents with the dynamic sequence instead of the static layout provided by compiler inspired many follow-up researches in processor architecture [11]. Various techniques that reduce power usage of trace caches have been proposed through selective trace cache lookup and update [12] or fetch direction predictor that predicts the next fetch from the trace cache or the instruction cache [13].

Trace cache designs are not popular in embedded processors due to the contradictory demands in power and throughput. Not using a sophisticated branch predictor to predict the trace at the front-end of an embedded processor, we demonstrated that the idea of trace delivery is still possible using the TR cache design. The TR cache delivers instructions in the form of traces like the trace cache, but with a much simpler management mechanism and lower complexity feasible for embedded processors.

C. Instruction Reuse

The other topic related to our work is the instruction reuse [31] proposed by Sodani in 1997. Early researches of instruction reuse have focused on avoiding the trivial computations to boost performance. Then the instruction recycling [32] is proposed to recycle the redundant trace in the SMT machine. To improve the ILP of superscalar processors, [33] and [34] proposed to apply instruction reuse in trace granularity. In 2006, Yang proposed a low-power trace reuse scheme [21] for the balanced performance between ILP and power. The mechanism proposed to identify reusable traces in [21] is similar to the TET lookup we proposed. However, the RIU in [21] employs a fully-associative structure and provides more sophisticated trace prediction and threading with the support of branch predictor.

A common feature of instruction reuse and trace reuse is the preserving and recycling of the decoded information. The ILP or power efficiency is improved via effectively buffering and reusing those repeated micro-operations, much like the decode filter cache [20] mentioned above. The TR cache we proposed does not attempt to reuse the repeated computation results or decode signals; only the instruction sequence preserved in the trace is extracted. Nevertheless, for processors that use complex out-of-order design, the number of reusable instructions in the reorder buffer may fluctuate since the pipeline utilization may change constantly. As a result, instruction reuse design in these processors tends to be complicated and not easy for extension. Moreover, the scalability of the instruction reuse schemes is mostly bounded by the instruction capacity of the pipeline. In the contrast, the TR cache is decoupled from the pipeline and hence provides a steady amount of reusable instructions and better scalability for extension.

D. Branch Resolution

Traditionally, the branch resolution includes two phases: one is the branch direction prediction and the other is the target address resolution. Algorithms for branch direction prediction have evolved from static and simple dynamic prediction reported in [1], [2] to complex schemes such as [3] for extremely high accuracy. Many of these prediction schemes incorporate a branch target buffer (BTB) as the means to acquire the predict-taken target addresses at the pipeline front-end. However, for the relatively simple pipeline of an embedded processor, the BTB itself imposes considerable power and area overhead. Petrov and Orailoglu proposed an application-specific BTB in [4] to eliminate such power and area overhead of conventional branch resolution schemes. The branch folding technique was also proposed in [5] to reduce the occurrences of branch instructions, which further reduce the requirement in the size of prediction logic. Salamat *et al.* also proposed to employ an adaptive hybrid direction prediction algorithm along with confidence-based pipeline gating in [6] as an alternative way to incorporate cost-effective branch resolution for embedded processors.

The TR cache combines the branch direction prediction and target address resolution in the process of trace reuse. Being an inherent feature of the trace-based instruction delivery mechanism, the branch resolution provided by the TR cache is solely hardware-based and does not require compiler support or profiling while [4] and [5] do. This offers better program binary compatibility and allows the TR cache to better serve as the option of improving an existing system. Moreover, the branch resolution schemes proposed in [4]–[6], though small in size, still impose power overheads on instruction delivery while the TR cache is able to reduce a significant portion of energy consumption in instruction delivery.

V. CONCLUSION

In this paper, the TR cache architecture is proposed as an alternative source for instruction delivery of embedded processors. The TR cache consists of a HTB to store the executed instructions and a TET to maintain a list of the trace-entry addresses and their positions in the HTB. By comparing the address of the incoming instruction with the trace-entry records in the TET, the processor is able to identify the reusable traces captured in the HTB. The processor can switch to the TR cache when a reusable trace is identified. In contrast with the conventional instruction cache which stores the pre-execution code with a static layout generated by the compiler, the TR cache preserves the post-execution program information in the form of dynamic instruction sequences. Such a post-execution cache offers the inherent capability of delivering traces without the support of complex trace-prediction and trace-construction hardware, therefore presents a feasible trace-based instruction delivery scheme for embedded processors. This trace-based delivery of instructions enables the TR cache to contribute branch prediction effect when it is activated. For an embedded processor with non-taken prediction scheme, the proposed TR cache is able to boost the prediction rate up to 92% along with 21% performance gain compared to the baseline processor. Moreover, due to the reduced hardware complexity, the TR cache is able to deliver instructions with lower energy costs than the conventional instruction cache. Experimental results indicate that the TR cache is able to provide the same level of energy-saving effect as the filter cache.

From the aspect of instruction delivery, the TR cache virtually expands the capacity of the conventional instruction cache. Our experiments showed that the TR cache is capable of sustaining the same level of IPC in a processor with downgraded instruction cache design. Specifically, the TR cache with 2048 entries is able to sustain 97% of the original IPC performance when the instruction cache capacity is significantly reduced from 16 to 2 kB. The capability of boosting performance and improving energy efficiency at the same time along with the use to partially substitute the function of a conventional instruction cache makes the TR cache a preferable choice among the design options of an embedded processor. Considering the overall area cost of the instruction delivery mechanism, our analysis concludes that adopting the TR cache can provide the performance gain of the branch predictor with the energy efficiency of the filter cache.

REFERENCES

[1] S. McFarling, "Combining branch predictors," Tech. Rep. Digital WRL, Jun. 1993.

[2] J. L. Hennessy and D. A. Patterson*, Computer Architecture: A Quantitative Approach*, 4th ed. San Francisco, CA: Kaufman, 2006.

[3] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proc. 7th Int. Symp. High-Perform. Comput. Arch.*, Jan. 2001, pp. 197–206.

[4] P. Petrov and A. Orailoglu, "Low-power branch target buffer for application-specific embedded processors," *IEE Proc. Comput. Digit. Techn.*, vol. 152, no. 4, pp. 482–488, Jul. 2005.

[5] P. Petrov and A. Orailoglu, "A reprogrammable customization framework for efficient branch resolution in embedded processors," in *Proc. ACM Trans. Embed. Comput. Syst.*, May 2005, pp. 452–468.

[6] B. Salamat, A. Baniasadi, and K. J. Deris, "Area-aware optimizations for resource constrained branch predictors exploited in embedded processors," in *Proc. Int. Conf. Embed. Comput. Syst.: Arch., Model. Simulation*, Jul. 2006, pp. 50–55.

[7] T. Y. Yeh and Y. N. Patt, "A comprehensive instruction fetch mechanism for a processor supporting speculative execution," in *Proc. 25th Annu. Int. Symp. Microarch.*, Dec. 1992, pp. 129–139.

[8] T. M. Conte, K. N. Menezes, P. M. Mills, and B. A. Patel, "Optimization of instruction fetch mechanisms for high issue rates," in *Proc. 22nd Int. Symp. Comput. Arch.*, May 1995, pp. 333–344.

[9] G. Reinman, T. Austin, and B. Calder, "A scalable front-end architecture for fast instruction delivery," in *Proc. 26th Int. Symp. Comput. Arch.*, May 1999, pp. 234–245.

[10] E. Rotenberg, S. Bennett, and J. E. Smith, "A trace cache microarchitecture and evaluation," *IEEE Trans. Comput.*, vol. 48, no. 2, pp. 111–120, Feb. 1999.

[11] A. Hossain, D. J. Pease, J. S. Burns, and N. Parveen, "Trace cache performance parameters," in *Proc. IEEE Int. Conf. Comput. Des.*, Feb. 2002, pp. 348–355.

[12] J. S. Hu, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir, "Selective trace cache: A low power and high performance fetch mechanism," Dept. Comput. Sci. Eng., Pennsylvania State Univ., Philadelphia, Tech-cse-02-016, 2002.

[13] J. S. Hu, N. Vijaykrishnan, M. J. Irwin, and M. Kandemir, "Using dynamic branch behavior for power-efficient instruction fetch," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Feb. 2003, p. 127.

[14] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, G. W. Hoeppner, D. Kruckemyer, T. H. Lee, C. M. Lin, L. Madden, D. Murray, M. H. Pearce, S. Santhanam, K. J. Snyder, R. Stephany, and S. C. Thierauf, "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *IEEE J. Solid-State Circuits*, vol. 31, no. 11, pp. 1703–1714, Nov. 1996.

[15] J. Kin, M. Gupta, and W. H. Magione-Simth, "Filter cache: An energy efficient memory structure," in *Proc. 30th Int. Symp. Microarch.*, Dec. 1997, pp. 184–193.

[16] J. Kin, M. Gupta, and W. H. Magione-Simth, "Filtering memory references to increase energy efficiency," *IEEE Trans. Comput.*, vol. 49, no. 1, pp. 1–15, Jan. 2000.

[17] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis, "Energy and performance improvements in microprocessor design using a loop cache," in *Proc. Int. Conf. Comput. Des.*, Oct. 1999, pp. 378–383.

[18] L. Lee, B. Moyer, and J. Arends, "Instruction fetch energy reduction using loop caches for embedded applications with small tight loops," in *Proc. Int. Symp. Low Power Electron. Des.*, 1999, pp. 267–269.

[19] W. Tang, R. Gupta, and A. Nicolau, "Design of a predictive filter cache for energy savings in high performance processor architectures," in *Proc. Int. Conf. Comput. Des.*, Sep. 2001, pp. 68–73.

[20] W. Tang, R. Gupta, and A. Nicolau, "Power savings in embedded processors through decode filter cache," in *Proc. Des. Autom. Test Euro. Conf. Exhibition*, Oct. 2002, pp. 443–448.

[21] C. Yang and A. Orailoglu, "Power-efficient instruction delivery through trace reuse," in *Proc. 15th Int. Conf. Parallel Arch. Compilation Techn.*, 2006, pp. 192–201.

[22] A. Efthymiou and J. D. Garside, "A CAM with mixed serial-parallel comparison for use in low energy caches," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 3, pp. 325–329, Mar. 2004.

[23] C.-H. Chen, C.-K. Wei, T.-H. Lu, and H.-W. Gao, "Software-based self-testing with multiple-level abstractions for soft processor cores," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 5, pp. 505–517, May 2007.

[24] T. Austin, E. Larson, and D. Ernst, "SimpleScalar, "An infrastructure for computer system modeling"," *IEEE Trans. Comput.*, vol. 35, no. 1, pp. 59–67, Feb. 2002.

[25] HP Labs, Palo Alto, CA, "CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model," 2009. [Online]. Available: http://www.hpl.hp.com/research/cacti/

[26] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," HP Laboratories, Palo Alto, CA, Tech. Rep. HPL-2008-20, Jun. 2006.

[27] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE 4th Annu. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.

[28] SEMATECH, Austin, TX, "International technology roadmap for semiconductors," 2005. [Online]. Available: http://www.itrs.net/

[29] J. Penton and S. Jalloq, "Cortex-R4: A mid-range processor for deeply-embedded applications," ARM white paper, May 2006. [Online]. Available: http://www.arm.com/products/CPUs/ARM_Cortex-R4F.html

[30] A. Janapsatya, S. Parameswaran, and A. Ignjatovic, "HitME: Low power hit memory buffer for embedded systems," in *Proc. Asia South Pacific Des. Autom. Conf.*, Jan. 2009, pp. 335–340.

[31] A. Sodani and G. S. Sohi, "Dynamic Instruction Reuse," in *Proc. 24th Annu. Int. Symp. Comput. Arch.*, Jun. 1997, pp. 194–205.

[32] S. Wallace and D. M. Tullsen, "Instruction recycling on a multiple-path processor," in *Proc. 5th Int. Symp. High-Perform. Comput. Arch.*, Jan. 1999, pp. 44–53.

[33] T. da Costa, F. M. G. Franca, and E. M. C. Filho, "The dynamic trace memorization reuse technique," in *Proc. Int. Conf. Parallel Arch. Compilation Techn.*, Oct. 2000, pp. 92–99.

[34] D. Charles, A. R. Hurson, and N. Vijaykrishnan, "Improving ILP with instruction-reuse cache hierarchy," in *Proc. 5th Int. Conf. Algorithms Arch. for Parallel Process.*, 2002, pp. 206–213.

Yi-Ying Tsai received the B.S., M.S., and Ph.D. degrees in electrical engineering from National Cheng Kung University, Taiwan, in 2001, 2003, and 2010, respectively.

His research interests include computer architecture, low-power methodology, software-based self testing, and binary translation.

Chung-Ho Chen (M'06) received the M.S.E.E. degree in electrical engineering from the University of Missouri-Rolla, Rolla, in 1989 and the Ph.D. degree in electrical engineering from the University of Washington, Seattle, in 1993.

In 1993, he was with the Department of Electronic Engineering, National Yunlin University of Science and Technology. In 1999, he joined the Department of Electrical Engineering, National Cheng-Kung University, Tainan City, Taiwan, where he is currently a Professor. His research areas include advanced computer architecture, graphics processing, and high-speed ESL simulation systems.

Prof. Chen was a recipient of the 2009 Outstanding Teaching Award from the National Cheng-Kung University. He was the Technical Program Chair of the 2002 VLSI Design/CAD Symposium held in Taiwan.