A Low-Power Instruction Cache Architecture Exploiting Program Execution Footprints

Inoue, Koji
Department of Computer Science and Communication Engineering, Faculty of Information Science and Electrical Engineering, Kyushu University

Murakami, Kazuaki
Department of Computer Science and Communication Engineering, Faculty of Information Science and Electrical Engineering, Kyushu University

出版情報：Seventh International Symposium on High-Performance Computer Architecture, 2001-01
バージョン：
権利関係：

A Low-Power Instruction Cache Architecture Exploiting Program Execution Footprints

Koji Inoue Kazuaki Murakami
Dept. of Computer Science and Comm. Eng. Kyushu University
6–1 Kasuga-koen, Kasuga, Fukuoka 816-8580 Japan.

1 Problem

On-chip caches have been playing an important role in achieving high performance processors. In particular, much higher performance is required for instruction caches because one or more instructions have to be issued on every clock cycle. In other words, from energy point of view, the instruction cache consumes a lot of energy. Therefore, it is strongly required to reduce the energy consumption for instruction-cache accesses.

In direct-mapped instruction caches, tag comparison and data read are performed in parallel. Thus, the total energy consumed for a cache access has two factors: the energy for the tag comparison and that for the data read.

Cache subbanking is one of approaches to reducing the data-access energy: the data-memory array is partitioned into several subbanks, and only the subbank which includes the desired data is activated [1]. We have calculated the energy consumption for a 16 KB direct-mapped cache based on Kamble's model [1]. Note that the energy for I/O drivers and address decoder is not included in this calculation. As a result, it has been observed that increasing the number of subbanks reduces a lot of energy for the data memory. Since the tag-access energy is maintained, however, the effect of the tag comparison becomes a significant factor on the total energy consumption. In fact, where the subbank width is the same as the processor-word width, the energy consumed for the tag-memory accesses occupies about 30 % and almost half of total energy in 32-bit and 64-bit microprocessors, respectively.

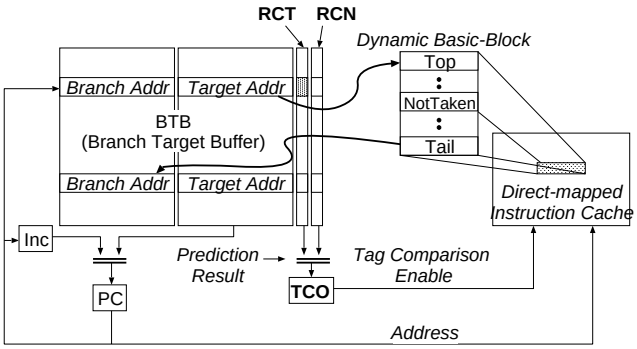
2 Solution

There are many loops in programs, so that some instruction blocks will be executed in many times. We call a run-time instruction block "*a dynamic basic-block*". The dynamic basic-block consists of one or more successive basic blocks. The top of the dynamic basic-block is addressed by a branch-target address, and the end of it is addressed by a taken-branch or jump address.

We consider where a dynamic basic-block is executed in many times during program execution. On the first execution of the dynamic basic-block, the tag comparison for all instructions has to be performed. However, on the second execution, if no cache miss has occurred since the first execution, it is guaranteed that the dynamic basic-block resides in the cache. At that time, we can determine that the indexed cache entry corresponds to the requested address without performing the tag comparison. We refer to the cache which attempts to omit the tag comparison by exploiting the execution history as *history-based tag-comparison cache*. When a dynamic basic-block is executed, the history-based tag-comparison cache attempts to avoid unnecessary tag comparisons by detecting the following conditions: 1) the dynamic basic-block has been executed, and 2) no cache miss has occurred since the previous execution of the dynamic basic-block.

To detect the conditions, execution footprints are recorded in a BTB (Branch Target Buffer). The footprint indicates whether the corresponding dynamic basic-block resides in the cache. If a dynamic basic-block left the footprint at the previous execution, then the tag comparisons for current execution can be omitted. All footprints are erased when a cache miss takes place, because a dynamic basic-block (or a part of the dynamic basic-block) might be evicted from the cache. In addition, the execution footprints are erased on a BTB replacement. Figure 1 depicts an organization of the extended BTB. The following flags are added to each BTB entry:

- RCT (Residing in Cache on Taken) 1-bit flag per entry : This is an execution footprint for a dynamic basic-block, the top of which is addressed by the corresponding target address. This flag is set to 1 when the corresponding branch is taken, and is reset to 0 whenever a cache miss or a BTB replacement occurs.



Benchmark	C-TC	IL-TC	H-TC	HIL-TC
099.go	1.000	0.3203	0.7604	0.2378
124.m88ksim	1.000	0.3302	0.4217	0.1361
129.compress	1.000	0.3528	0.1751	0.0706
126.gcc	1.000	0.3343	0.6810	0.2278
130.li	1.000	0.3515	0.4500	0.1684
132.ijpeg	1.000	0.2992	0.1062	0.0311
134.perl	1.000	0.3436	0.6643	0.2249
147.vortex	1.000	0.3213	0.8838	0.2837
102.swim	1.000	0.2957	0.0623	0.0278
107.mgrid	1.000	0.2600	0.0008	0.0002
110.applu	1.000	0.2657	0.0252	0.0070
125.turb3d	1.000	0.2813	0.0849	0.0266
141.apsi	1.000	0.2801	0.1050	0.0307

- RCN (Residing in Cache on Not-taken) 1-bit flag per entry : This is an execution footprint for fall-though instructions. This flag is set to 1 when the corresponding branch is not-taken, and is reset to 0 whenever a cache miss or a BTB replacement occurs.

In addition, a flag to enable the tag comparison in the cache is required:

- TCO (Tag-Comparison Omit) 1-bit flag : This flag indicates whether the tag comparison can be omitted. If this flag is 1, the tag comparison is not performed. On every BTB hit, the corresponding execution footprints (i.e., the RCT flag or the RCN flag) is stored to the TCO. Which execution footprint has to be selected depends on the branch-prediction result (the RCT flag on taken, and the RCN flag on not-taken). The TCO flag is reset to 0 whenever a cache miss or a BTB replacement occurs.

Since the TCO is not on cache critical-paths, there is no cache-access-time overhead in the history-based tag-comparison cache.

3 Evaluations

We have measured the total count of tag comparison required for program execution. The SPEC CPU benchmark programs are executed using the SimpleScalar simulator [3]. We have modified the simulator to implement the history-based tag-comparison cache. In this simulation, the following configuration is assumed: cache size is 32 K bytes, cache-line size is 32 bytes, the number of direct-mapped BPT entry is 2048, the number of BTB set is 512, the BTB associativity is 4, the RAS size is 8.

Table 1 shows the simulation results. All results for each program are normalized to the conventional tag-comparison cache (C-TC), in which the tag comparison is performed on every access. The interline tag-comparison cache (IL-TC), which is another approach to omitting the tag comparison [2], performs the tag comparison only when two successive instructions reside in the same cache line (the tag comparison for the later instruction is omitted). The combination of the IL-TC and our history-based tag-comparison cache (H-TC) is also evaluated, which is denoted as HIL-TC (history-based interline tag-comparison cache).

Since there are many incremental accesses in almost all programs, IL-TC works well for all programs. While the effectiveness of the realistic history-based tag-comparison cache (H-TC) is application dependent, because the cache exploits iterative execution in programs. H-TC produces better results than IL-TC for two integer programs, *129.compress* and *132.ijpeg*, and for all floating-point programs. In particular, the cache reduces almost equal to or more than 90 % tag comparisons for the floating-point programs, *102.swim*, *107.mgrid*, *110.applu*, *125.turb3d*, *141.apsi*. In addition, it can be seen from the simulation results that the combination of our history-based approach and the interline approach makes a significant reduction for the tag comparison.

References

[1] Kamble M. B., and Ghose K., "Analytical Energy Dissipation Models For Low Power Caches," *Proc. of the 1997 International Symposium on Low Power Electronics and Design*, pp.143–148, Aug. 1997.

[2] Panwar R., and Rennels D., "Reducing The Frequency of Tag Compares for Low Power I-cache Design," *Proc. of the 1995 International Symposium on Low Power Electronics and Design*, pp.57–62, Apr. 1995.

[3] "SimpleScalar Simulation Tools for Microprocessor and System Evaluation," *URL:http://www.simplescalar.org/*.