Trace cache

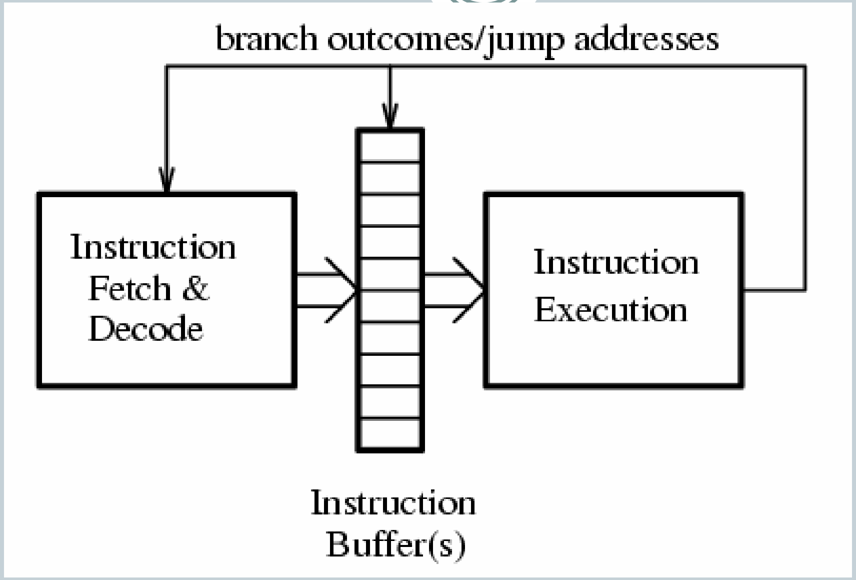
ADVANCED COMPUTER ARCHITECTURE
UNIVERSITY OF JORDAN
DR. GHEITH ABANDAH

DONE BY
DUA'A AL-NAJDAWI

Introduction I:

- Now all the processors are superscalar processors and have the instruction level parallelisms technique. The superscalar design has to increase the scale of these techniques: wider dispatch/issue, larger windows, more physical registers, more functional units, and deeper speculation. As the issue width of superscalar processors is increased, instruction fetch bandwidth requirements will also increase

Introduction II:



- instruction fetch bandwidth is becoming a performance bottleneck. There are also some other factors appearing when issue rate exceeds four instructions per cycle:

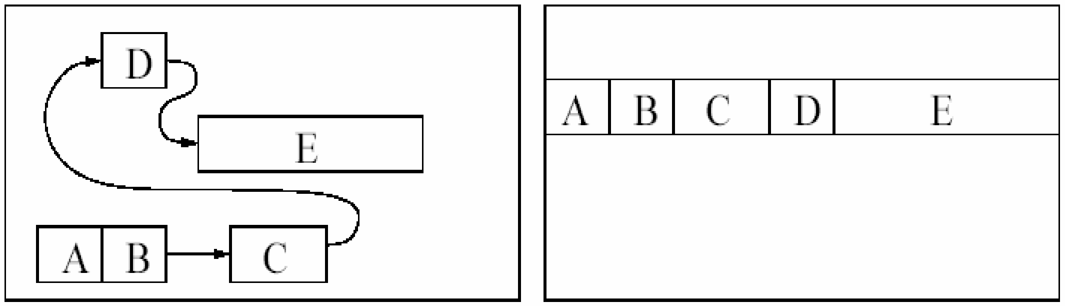
- (1) Branch throughput: if only one condition branch is predicted per cycle, then the window can grow at the rate of only one basic block per cycle.
- (2) Noncontiguous instruction alignment: instructions to be fetched may not be in contiguous cache locations because of conditional branch or jump instructions, so it will cause a high fetching latency if there is no other logic to fetch these instructions in parallel and align them and pass them up the pipeline.
- (3) Fetch unit latency: if a branch was mispredicted, recovery much be done to resolve this mispredition. Some instructions have to be squashed from pipeline stages and fetching needs to be redirected. The startup cost of redirecting fetching will cause fetch unit latency.

- Trace Cache technique was proposed by Rotenberg et al [1] to address the above problems.
- **"Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching."**

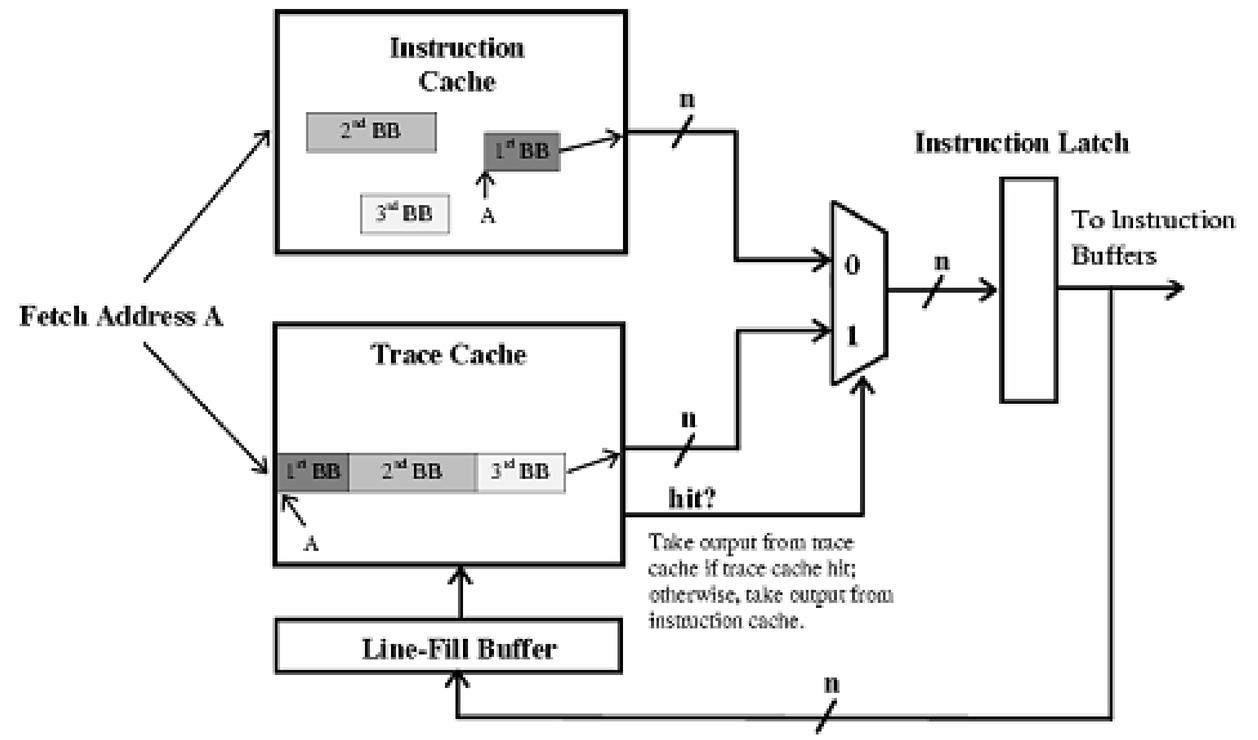
- For the Instruction Benchmark Suite (IBS) and SPEC92 integer benchmarks, a 4 kilobyte trace cache improves performance on average by 28% over conventional sequential fetching.

What is trace cach?

- Trace Cache is a hardware structure, each line of which stores a snapshot, or trace, of dynamic instruction stream. A trace is a sequence of at most n instructions and at most m basic blocks starting at any point in the dynamic instruction stream.



(a) Instruction cache. (b) Trace cache.


Instruction Cache
2nd BB
1st BB
3rd BB
A
n
Instruction Latch
0
1
n
To Instruction Buffers
Fetch Address A
Trace Cache
1st BB
2nd BB
3rd BB
A
n
hit?
Take output from trace cache if trace cache hit; otherwise, take output from instruction cache.
Line-Fill Buffer
n

Result

- The trace cache is a proposed solution to achieving high instruction fetches bandwidth by buffering and reusing dynamic instruction traces. This work presents a new block-based trace cache implementation that can achieve higher IPC performance with more efficient storage of traces.

Trace Cache Sampling Filter

- This is new technique suggests that instead of building all the traces and trying to select the good ones among them, it is more efficient to make a preliminary selection of traces. This selection is based on a random sampling approach.

Trace Cache Sampling Filter

- the Sampling Filter improves trace cache and overall system performance, while reducing power dissipation. The Sampling Filter reduces admission of traces that are not used prior to their eviction from the cache, and prolongs the percentage of time a trace is in its live phase during its stay in the cache. Moreover, the Sampling Filter reduces duplication between the trace cache and the instruction cache and thus reduces the overall misses in the first level of cache hierarchy.

Trace Cache Sampling Filter

- I is simple mechanism to increase the utilization of a small trace cache, and simultaneously reduce its power consumption. The sampling filter exploits the “hot/cold trace” principle, which divides the population of traces into two groups.

Trace Cache Sampling Filter

- The first group contains “hot traces” that are executed many times from the trace cache and contribute the majority of committed instructions. The second group contains “cold traces” that are rarely executed, but are responsible for the majority of writes to an unfiltered cache.

Trace Cache Sampling Filter

- The sampling filter selects traces without any prior knowledge of their quality. However, as most writes to the cache are of "cold traces" it statistically filters out those traces, reducing cache turnover and eventually leading to higher quality traces residing in the cache.

Trace Cache Sampling Filter

- Results show that the sampling filter can increase the number of hits per build (utilization) by a factor of 38, reduce the miss rate by 20% and improve the performance-power efficiency by 15%.Further improvements can be obtained by extensions to the basic sampling filter: allowing “hot traces” to bypass the sampling filter, combining of sampling together with previously proposed filters, and changing the replacement policy in the trace cache. Those techniques combined with the sampling filter can reduce the miss rate of the trace cache by up to 40%.

Thank you