TCPT - Thread Criticality-driven Prefetcher Throttling

Biswabandan Panda
Dept. of CSE, Indian Institute of Technology, Madras
Email: biswa@cse.iitm.ac.in

Shankar Balachandran
Dept. of CSE, Indian Institute of Technology, Madras
Email: shankar@cse.iitm.ac.in

***Abstract*—A single parallel application running on a multicore system shows sub-linear speedup because of slow progress of one or more threads known as critical threads. Identifying critical threads and accelerating them can improve system performance. One of the metrics that correlate to thread criticality is the number of cache misses and the penalty associated with it. This paper proposes a throttling mechanism called *TCPT* which throttles hardware prefetchers by changing the prefetch degree based on the thread criticality.**

Motivation: Prefetcher throttling technique like HPAC [2] has been explored for sequential applications running on muticore systems. HPAC coordinates and controls the aggressiveness of multiple prefetchers using prefetch metrics. HPAC [2] does well for sequential applications. However, in case of a single parallel application, the effectiveness of HPAC [2] is marginal. Per core prefetch metrics like accuracy and DRAM bandwidth consumption does not consider the constructive interference among threads of a single parallel application. HPAC [2] does consider inter core pollution but for applications with high degree of sharing, it is nullified by the constructive interference among threads. We propose a hardware prefetcher throttling technique called *T*CPT which controls the aggressiveness of the prefetcher based on the progress of the threads and the number of prefetches issued (PF_{issued}). Aggressiveness prefetching makes the critical threads faster that causes reduction in the number of cache misses.

TC, Slack and TP: We modify a recently proposed thread criticality predictor [1], which accounts for L1 and L2 cache misses and the miss penalty associated with it. We extend the predictor for a three level cache hierarchy which also accounts for demand responses from remote cores. `Thread Criticality` (*T*C) of thread i is defined as

$$TC(i) = L2_{hits}(i) + L_i * L3_{hits}(i) + R_i * Remote_{hits}(i) + LL_i * DRAM_{hits}(i)$$

where, $R_i = \frac{remotepenalty}{L1penalty}$, $L_i = \frac{L1L2penalty}{L1penalty}$, and $LL_i = \frac{L1L2L3penalty}{L1penalty}$

$L2_{hits}(i)$ is the number of L1 misses that hit in the local L2. $L3_{hits}(i)$ is the number of L1 misses that also miss in L2 and hit in the shared L3. $Remote_{hits}(i)$ is the number of L1 or L2 misses that hit in the L1 or L2 of the remote cores. $DRAM_{hits}(i)$ is the number of L1 misses that miss in L2 and L3 and hit in the DRAM. We use a metric called `Slack` which tracks the difference between speed of the threads. For a given thread i, $Slack(i) = max(TC(j))_{\forall j=0\ to\ n-1} - TC(i)$, where n is the total number of threads in an application.
The thread with $slack = 0$ is treated as the critical(slowest) thread. We also use a metric called `Normalized Thread Criticality`(*N*TC) of thread i which we define as $NTC(i) = TC(i)/LS(i)$ where, $LS(i)$ is the number of Loads and Stores that miss in L1. We use a $Slack_{threshold}$ register to find multiple critical threads, if any. Threads with slack value less than $Slack_{threshold}$ are treated as critical threads. The objective of *TCPT* is to speedup the critical threads which are prefetch friendly. To check whether a thread i, is prefetch friendly or not, we define a metric called `Thread Progress`(*T*P). *T*P of a given thread i at a given simulation window j as $TP_{ij} = NTC(i)_{j-1} - NTC(i)_j$. If TP_{ij} is positive then the thread i is progressing.

TCPT: We use `TP(i)`, `Slack(i)` and $PF_{issued}(i)$ of a thread i to throttle a prefetcher. We place *TCPT* beside the shared L3 cache which collect these metrics at a regular interval of 100K cycles and resets it to zero after making a decision. Each thread maintains a set of counters to calculate the relevant metrics and a $lastTC_i$ register which stores the `TC` value of the previous simulation window. Table I shows the throttling decisions in *TCPT* which are based on the progress of a thread.

TABLE I. DECISIONS BASED ON TCPT

$Critical?$	$Progress?$	PF_{issued}	Throttling	Rationale
Yes	Yes	-	degree=4	minimize criticality
Yes	No	Low	degree=2	prefetcher unfriendly
Yes	No	High	degree=1	minimize pollution
No	Yes	Low	degree=4	prefetcher friendly
No	Yes	High	degree=2	minimize pollution
No	No	-	degree=1	minimize pollution

Results: We use gem5 FS simulator to simulate a three level cache hierarchy. Our baseline system setup has a shared L3 cache with HPAC controlling the per core stride prefetchers which prefetches into L3 from DRAM. We compare *TCPT* with the baseline prefetcher in terms of execution time. On an average, *TCPT* beats the baseline prefetcher by 8.13% for 4 cores. Benchmarks like `streamcluster` and `vips` show improvement of 13.4% and 17%. In case of 8 cores, *TCPT* improves the execution time by 7% as compared to the baseline prefetcher. For applications, where all the threads progress with the same speed, the effectiveness of *TCPT* is marginal(less than 1%) . We observe this trend in applications like `blackscholes` and `canneal`.

REFERENCES

[1] Bhattacharjee et al.,*"Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors"*, ISCA 2009, pp. 290-301

[2] Ebrahimi et al., *"Coordinated control of multiple prefetchers in multi-core systems"*, MICRO 2009, pp. 316-326

978 -1-4799-1021-2/13/$31.00 ©2013 IEEE