DDC-I

WHITE PAPER

Cache Partitioning Improves Cache Performance for Safety-Critical Applications

A big challenge facing developers of safety-critical software applications is managing contention for shared resources such as cache. Benchmarks demonstrate that cache contention can increase worst-case execution times (WCETs) up to 4x higher than average-case execution times (ACETs). Unless developers can bound and control these WCETs, processor utilization is seriously diminished and analysis of inter-application interference patterns greatly complicate safety certification. One way that developers can effectively manage this contention and boost available CPU time to time critical tasks is to utilize DDC-I's cache partitioning (patent pending).

Worst-case execution

All applications compete for shared cache (L2, and/or L3 if present). This greatly increases the potential for interference whereby the worst case execution time of a safety critical application is significantly impacted by another application that is memory intensive and constantly overruns the cache, causing cache misses for the safety critical program. Consequently, the impacted software may overrun its execution time budget and/or miss deadlines, resulting in unsafe failure conditions.

Processors are designed to optimize average-case execution times (ACETs), often at the expense of worst-case execution times (WCETs). But while optimized ACETs work well in non-critical applications, developers of certifiable, safety-critical software must design and budget application execution times for WCET behavior. Consequently, applications require significantly higher execution time budgets (much of it often unused), resulting in significantly degraded CPU time budget available for time critical tasks in the system as a whole.

Cache partitioning reduces WCET and increases CPU utilization by reducing cache competition and making it easier to bound and control interference patterns. By setting aside dedicated partitions for critical applications, developers can reduce interference from applications running in the system and provide timely, deterministic access to cache. This reduces the amount time that must be budgeted for critical tasks, thereby shrinking the delta between ACET and WCET and boosting overall CPU utilization.

Cache Partitioning

In a typical processor configuration the CPU has a small cache (10’s of KB) as the L1 cache and then a larger L2 cache (100’s of KB to a few MB). With this configuration, the L1 cache is small and closely coupled to the larger L2 cache; therefore its cache effects are negligible in the system impact analysis. The main impact on system performance is contention in the L2 cache.

In this configuration, all applications executing in the system compete for the entire L2 cache in normal operation. If application A on Core 0 uses data that maps to the same cache line(s) as application B, then a collision occurs.

For example, suppose A’s data resides in L2; access to that data will take very few processor cycles. Then, suppose B accesses data that happens to map to the same L2 cache line as A’s data. At that point, A’s data must be evicted from L2 (including a potential “write-back” to RAM), and B’s data must be brought into cache from RAM. The time required to handle this collision is typically charged to B. Then, suppose A accesses its data again. Since that data is no longer in L2 (B’s data is in its place), B’s data must be evicted from L2 (including a potential “write-back” to RAM), and A’s data must be brought back into cache from RAM. The time required to handle this collision is typically charged to A.

Most times, A and B will encounter such collisions infrequently. In those cases, their respective execution times can be considered as “average case” (i.e., ACETs). However, on occasion, their data accesses will collide at a high frequency. In these cases, their respective execution times must be considered as “worst case” (i.e., WCETs).

When developing certifiable, safety-critical software, one must design and budget an application’s execution time for worst-case behavior, since such software must have adequate time budget to complete its intended function every time it executes, lest it cause an unsafe failure condition (note that a safety-critical RTOS must enforce time partitioning, wherein each application has a fixed amount of CPU time budget to execute).

Cache partitioning reduces WCET by reducing cache collisions among competing applications. This partitioning eliminates the possibility of applications interfering with one another via L2 collisions. Without such interference, the deltas between application WCETs and its ACETs often are often considerably lower than is the case without cache partitioning. By bounding and controlling these interference patterns, application execution times are more deterministic and time budgets can be of shorter duration, thereby keeping processor utilization high.

Test Environment and Applications

In the following test cases we will show how cache partitioning improves the WCETs on a single core processor. In the future, as multicore processors become more widely used in safety critical environments, the benefits of cache partitioning will become amplified as multiple applications on multiple cores can start interfering with each other at any time during normal execution due to collisions in the L2 cache.

Cache partitioning tests were performed on a 1.6-GHz Atom processor (x86) with 32KB of L1 data cache, 24KB of L1 instruction cache, and a 512KB unified L2 cache. A single core x86 processor was used for these tests. Cache partitioning capability delivers benefits to any applications competing for L2 cache. Additionally, it does not depend on any features that are special or unique to x86 processors and applies equally well to other processor types (such as ARM or PowerPC).

Four memory-intensive test applications were used, all using a range of data/code sizes, sequential and random access strategies, and various working set sizes:

- read-only
- write-only
- copy
- code execution

Tests were run with and without a “cache trasher” application, which evicts test application data/code from L2 and “dirties” L2 with its own data/code. In effect, the cache trasher puts L2 into a worst-case state from a test application’s perspective. That is, the cache trasher mimics real-world scenarios, where different applications run concurrently and compete for the shared L2 cache.

Each test application was executed under three scenarios:

In scenario 1, without cache partitioning and without cache trashing, the test application competes for the entire 512KB L2 along with the RTOS kernel and a variety of debug tools. This test establishes baseline average performance, wherein each test executes with an “average” amount of L2 contention.

In scenario 2, without cache partitioning and with cache trashing, the test application competes for the entire 512KB L2 along with the RTOS kernel, a variety of debug tools and the rogue cache trasher application. This test establishes baseline worst-case performance, wherein each test executes with a worst-case amount of L2 interference from other applications, primarily the cache trasher.

In Scenario 3, with cache partitioning and with cache trashing, three L2 partitions are created: a 256KB partition allocated to the test application; a 64KB partition allocated to the RTOS kernel and a variety of debug tools; and a 192KB partition allocated to the rogue cache trasher application. This scenario establishes optimized worst-case performance, wherein each test executes within its own L2 partition with no interference from other applications, including the cache trasher.

Results and Benefits of Cache Partitioning

Results of the read-only test application demonstrate the benefits of cache partitioning. These results are representative of the other three tests.

With no cache partitioning and no cache trashing (scenario 1, ACET), the read-only test averaged 105 usecs to execute given a working set size of 512KB. With no cache partitioning, but with cache trashing (scenario 2, WCET1), the test took roughly 400 usecs to execute given the same size working set (a 280% increase). However, with cache partitioning and cache trashing (scenario 3, WCET2), the average execution time drops back to 117 usecs, or just 11% higher than the ACET.

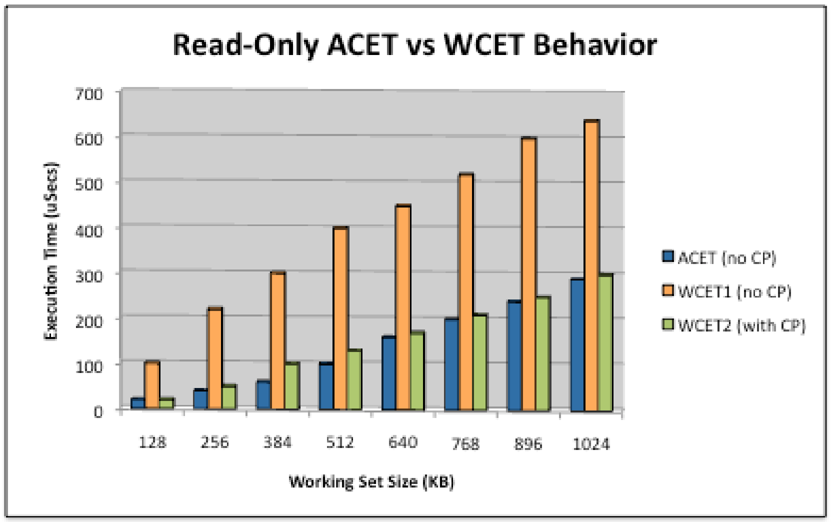

Figure 1 – Cache Partitioning Impact on Read-Only Tests

These results clearly demonstrate the efficacy of cache partitioning for an application that performs a large number of reads per period. Though difficult to discern here, the impact on bounding WCETs is more pronounced when the application's working set size fits within the cache partition that it's using (in this case, 256KB). This result is expected due to the nature of cache. That said, embedded, real-time applications tend to have relatively small working set sizes, in these cases cache partitioning will benefit most applications.

Results for the write-only test were similar to the read-only test, though more pronounced for smaller working sets. For larger working sets, results showed relatively small differences between WCETs with and without cache partitioning. Results for the copy test were similar to the read-only test, though more pronounced for smaller working sets. For larger working sets, results were less dramatic, but still showed significant improvement (roughly 2x) in WCETs with cache partitioning.

Note that it is possible for applications executing in the same cache partition to interfere with each other. However, such interference is much easier to analyze and bound than the unpredictable interference patterns that may occur between applications executing on different cores with shared cache. In those situations, applications could be mapped to separate cache partitions.

The benchmark results clearly demonstrate that cache partitioning technology provides an effective means of bounding and controlling interference patterns in cache for safety-critical time-partitioned applications. In particular, worst-case execution times are bounded and controlled much more tightly when the cache is partitioned. Consequently, application developers can set relatively tight, yet safe, execution time budgets thereby maximizing processor utilization.

For Additional Information