Pull based Migration of Real-Time Tasks in Multi-Core Processors

1. Problem Description

The complexity of uniprocessor design attempting to extract instruction level parallelism has pushed the computer architects to leverage parallelism through multiple simple cores on a single chip. Also, with continuous advancement in chip technology chip multi-processors (CMP) have become a reality. Multicores are becoming ubiquitous, not only in general-purpose but also embedded computing. However, on such platforms prediction of timing behavior of real-time tasks is becoming increasingly difficult. While real-time multicore scheduling approaches help to assure deadlines based on firm theoretical properties, their reliance on task migration poses a significant challenge to timing predictability in practice. Task migration actually (a) reduces timing predictability for contemporary multicores due to cache warm-up overheads while (b) increasing traffic on the network-on-chip (NoC) interconnect.

2. Related Work

Real-time tasks are usually periodic in nature and have to be completed before a predefined deadline. Missing a deadline could have serious consequences for hard real time systems. Recent work has shown that the impact of task migration could lead to increase in the execution time starting from 1% percent to 56% [1]. However, in that work a push model has been discussed that modifies the contemporary micro-architecture to enable the cache controller of source core, where the task is currently running and will stop execution, to migrate valid cache lines of the task to the target core, where the task will resume execution. This work overlaps the slack time between subsequent executions of the task on two different cores with migration of valid cache lines such that the target cache is already warmed up before the task starts executing on the target. This prevents the cache warm up from increasing the execution time of the migrated task.The primary disadvantage of the push model is that the contemporary architecture is incapable of pushing the cache lines from source core to target. Hence, the push model requires significant change in the micro-architecture.

3. Pull Model Design

We propose to develop a pull model to migrate the cache lines of the migrated task through memory read requests posted the target instead of a push request.
Our experimental model is a SMP based architecture. This choice is made so that the design can exhibit properties similar to the contemporary Tile-based [2,6] architecture minus interconnects and directory. It then excludes the complexity introduced by interconnects and uncovers the predictability challenge caused by cache misses only. So, the simulated environment will be a CMP, where each core is a SMT processor[3] that can run two contexts simultaneously. Since such cores are already present, a complete software solution will be one where the scheduler activates a pre-fetching thread at the target as soon as it decides to migrate a task. This pre-fetching thread can run independently of the task that is currently executing on target. This pre-fetcher thread may get the information about the critical regions of the task from the RTOS which it can then use to migrate cache lines. However, contexts running on SMTs have been known to

contend for all the critical resources on a core like the fetch stage and load store queues. Therefore, a pre-fetcher thread may induce unpredictability of execution time of the concurrent task running on the target. Thus, we propose a microarchitectural design that includes a dedicated hardware pre-fetcher that gets activated by the scheduler when it makes the decision of migrating a task. The pre-fetcher gets the information from the RTOS about the critical regions of the task. This pre-fetcher will not contend for the resources within the processor pipeline but at the memory hierarchy level. However, the study of increase in execution time experienced by the concurrently running tasks due to contention at memory hierarchy is out of scope of this work.

4. Infrastructure

This project involves microarchitectural modifications. Thus, we will use SESC simulator [4] to design the system. We will use WCET benchmarks from Malerdalen for testing the correctness of our modifications and effectiveness of our model.

5. Milestones

Week 1 & 2: Modify the Simulator such that it can allow a thread running on a separate core to migrate a task from on any source core to any target core. This will allow a scheduler to run on a separate core and cause the migrations to occur.

Week 3 & 4: Implement the hardware prefetcher that gets the information from the scheduler about the critical regions of a task. It starts pre-fetching the cache lines sequentially from the specified regions.

Week 5 & 6: Port the WCET benchmarks from Malardalen with the constructed infrastructure. Obtain the results for Pull migration scheme.

6. References

[1] A. Sarkar, F. Mueller, H. Ramaprasad, S. Mohan. Push-Assisted Migration of Real-Time Tasks in Multi-Core Processors. To appear in Proceedings of the 2009 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'09), Dublin, Ireland, June 19-20, 2009.
[2] M. Zhang and K. Asanovic. Victim migration: Dynamically adapting between private and shared cmp caches. TR 2005-064, MIT CSAIL,2005.
[3] Simultaneous Multithreading: Maximizing On-Chip Parallelism, D.M. Tullsen, S.J. Eggers, and H.M. Levy, In 22nd Annual International Symposium on Computer Architecture, June, 1995
[4] J. Renau, B. Fragela, J. Tuck, W. Liu, L. Ceze, S. Sarangi, P. Sack, and a. P. M. K. Strauss. Sesc simulator. http://sesc.sourceforge.net, Jan. 2005.
[5] Mälardalen benchmarksuite. http://www.mrtc.mdh.se/projects/wcet/benchmarks.html
[6] Tilera processor family. http://www.tilera.com/products/processors.php.

Project URL : http://www4.ncsu.edu/~asarkar/CSC714/home.html

Submitted by: Abhik Sarkar
Unity ID: asarkar