Policies for Migration of Real-Time Tasks in Embedded Multi-Core Systems

Kedar M. Katre[1], Harini Ramaprasad[1], Abhik Sarkar[2], Frank Mueller[2]
kedarked@siu.edu, harinir@siu.edu, asarkar@ncsu.edu, mueller@cs.ncsu.edu
[1]*Southern Illinois University Carbondale,* [2]*North Carolina State University*

Abstract

The increasing computational and power demands of embedded systems today are being met by deploying multi-core architectures. Several embedded systems have real-time requirements that necessitate offline temporal guarantees. The use of multicores in such systems poses a challenge in terms of timing predictability, specifically when real-time tasks are permitted to migrate among the different cores.

The aim of this paper is to put forth novel policies to guide migration decisions on time-critical and safety-critical embedded systems that use multicore architectures. Migration decisions are based on the cache usage of tasks, the migration mechanisms available and the characteristics of the network-on-chip (NoC) that is used to provide communication among cores.

1. Introduction

Increasing computational demands over the years have been addressed by increasing the operating clock frequencies of microprocessors as required. However, these designs have reached a clock frequency wall due to area and power considerations, leading to designs with multiple processors on a single chip, known as chip multiprocessors (CMPs) or simply *multicore* processors. The invention of multicore processors has, to a great extent, ensured that the rate of increase in performance of computing systems is maintained, thereby making multicores ubiquitous these days.

The electronics industry has been experiencing an upsurge with the advent of embedded systems. Embedded systems have been a major contributor in reducing the cost, power and area requirements of computing systems. Until recently, embedded systems worked on single-core microprocessors. However, due to increasing computational demands even on such systems, multicore architectures have already found their place in the embedded systems domain.

Prediction of timing behavior to ensure that real-time task deadlines are met is becoming increasingly difficult with the use of multicore platforms in embedded systems. While several real-time multicore scheduling strategies have been and are being proposed to address this issue, their reliance on task migration remains a major challenge. Task migration among cores reduces timing predictability due to cache warm-up overheads while increasing traffic on the Network-on-Chip (NoC) interconnect.

This work was supported in part by NSF grants CNS-0905212, CNS-0905181 and CNS-0720496.

In this paper, we present novel policies to guide migration decisions on embedded multicore systems that require temporal guarantees. Migration decisions are based on cache usage of tasks, the migration mechanisms available and characteristics of the NoC used for communication among cores. We assume that a task may be migrated only between jobs, or, in other words, at the end of the execution of one job and before the next job begins. The core that a task is executing on just before it is migrated is called the *source* core for the migration and the core that the task is migrated to is called the *target* core.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents relevant background information and motivates the problem further. Section 4 describes the methodology used to make migration decisions. Finally, we present our conclusions in Section 5.

2. Related Work

Choffnes *et al*. propose migration policies for multicore fair-share scheduling in the context of soft real-time systems [8]. Their technique minimizes migration costs while ensuring fairness among tasks by maintaining balanced scheduling queues as new tasks are activated. In contrast, our work targets hard real-time systems.

Li *et al*. present migration policies that facilitate efficient operating system scheduling in asymmetric multicore architectures [11], [12]. Their work focuses on fault-and-migrate techniques to handle resource-related faults in heterogeneous cores and does not operate in the context of real-time systems. In contrast, our work focuses on homogeneous cores and strives to improve system utilization by allowing migrations while providing timeliness guarantees for real-time systems.

Yan and Zhang have proposed techniques to calculate the worst-case execution time (WCET) of real-time tasks executing on multicores [19], [18], [20]. Other approaches develop WCET and cache analysis techniques for multi-level caches [14], [10]. All these approaches are limited to shared L2 instruction caches in WCET calculation and they do not consider task migration.

Calandrino *et al*. propose scheduling techniques to account for co-schedulability of tasks with respect to cache behavior [1], [7]. Their approach organizes tasks with the same period into groups of cooperating tasks. While their method improves cache performance in soft real-time systems, they do not specifically address issues related to task migration.

Ramaprasad *et al*. propose techniques to bound the data cache-related preemption delay (D-CRPD) of tasks in the context of periodic, hard real-time systems [15]. We use the results of this work for offline calculation of migration delay bounds (Section 4.1).

3. Motivation and Prior Work

In this section, we present some background information related to task scheduling on multicore systems and further motivate a systematic assessment of migration policies in real-time multicore systems.

3.1. Multicore Scheduling

Scheduling of tasks on cores is an important factor of consideration on multicore systems. Researchers have proposed several schemes for scheduling tasks on multicore systems. They may be broadly classified as partitioned and global scheduling policies.

In *partitioned scheduling* ([9], [6]), tasks are assigned to cores statically and are not allowed to migrate between cores. The advantage of using partitioned scheduling is that there is no migration overhead. However, partitioned scheduling suffers from two main disadvantages. First, such schemes are inflexible and cannot easily accommodate dynamic tasks without a complete re-partition. The re-partitioning problem may be resolved by allocating incoming dynamic tasks to the first available core, but this may not be optimal in terms of overall system utilization. Second, optimal assignment of tasks to cores is an NP-hard problem for which polynomial-time solutions result in sub-optimal partitions.

In *global scheduling* policies, tasks are allowed to migrate between cores as required. Recently, several optimal global scheduling policies have been proposed ([5], [13], [2], [3], [17], [4]). While these schemes strive to overcome the limitations of partitioned scheduling, they add migration overheads to tasks. In the context of real-time systems, the addition of migration overheads changes the timing behavior of tasks, thereby affecting the timing predictability of the system. This necessitates the incorporation of task migration overheads in analysis techniques, thus providing the motivation for, and demonstrating the importance of, the work presented in this paper.

3.2. Reasons for Migration

As discussed in Section 3.1, global real-time scheduling policies on multicores permit task migration among cores. There may be several reasons to do this despite the fact that migration introduces overheads on task execution time. Some of the reasons are listed below.

Capitalize on early task completion: Actual execution time of a real-time task is often significantly less than the worst-case execution time estimate. Early completion of a task on a particular core may be used to an advantage by migrating waiting tasks on a busy core to the newly idle core. This enables earlier start/resumption of the waiting task and improves utilization by minimizing idle time.

Facilitate aperiodic job admission: Task migration may be used to increase admissibility of sporadic jobs (aperiodic jobs with hard deadlines) into the system and to improve response times of aperiodic jobs with soft deadlines.

Avoid costly preemptions: A task potentially preempting a lower-priority task executing on a particular core may be migrated to a different core to allow significant reduction in the preemption delay that would otherwise be incurred by the lower-priority task.

Improve cache performance: If two or more tasks scheduled on the same core overlap significantly in their cache footprint, one or more of them may be migrated to a different core to reduce cache interference.

Balance load, power and thermal characteristics: Task migration may be used to balance the load on cores to ensure that no single core gets overheated while another core is idle.

3.3. Migration Mechanisms

Architectural and hardware support for actual migration may be provided in different ways. In prior work [16], we presented several mechanisms to facilitate task migration among cores. The basics of the mechanisms we use in the current work are described below.

3.3.1. Pull-based model (Conventional warm-up). No specific support for migration is provided in this case. Once the migrated task starts executing on the target core, any cache accesses that result in misses are resolved one at a time using the coherence protocol in effect within the system, either from the shared $L3$ cache or from the $L1/L2$ caches of the source core.

3.3.2. Push-based model. In this scheme, cache lines of the task to be migrated are proactively pushed from the source core cache to the target core cache. We currently consider two push models, as described below.

Whole Cache Migration: In this scheme, every line of the source core cache is consulted to identify lines belonging to the task to be migrated that have to be pushed to the target core.

Regional Cache Migration: In this scheme, programmers are allowed to define regions that correspond to a particular cache and only these regions are considered while migrating cache lines.

4. Methodology

In this section, we describe the methodology used to determine when and what task to migrate and where to migrate the task to. For this purpose, we develop a cost-benefit analysis technique that considers several factors to determine the feasibility and usefulness of a given migration.

4.1. Offline Analysis: Migration Delay Bounds

An offline component is employed for calculating the worst-case possible delays introduced into the system due to task migrations. Migration delay bounds include:

- Worst-case Migration Related Preemption Delay (MRPD) experienced by the tasks on the source core due to migration;
- Worst-case MRPD experienced by the existing tasks on the target core due to migration;
- Worst-case Migration Related Cache Delay (MRCD) experienced by the migrated task;
- Worst-case Communication Delay (WCCD) between the source and target cores.

In order to calculate the first three of the four bounds above, we employ static analysis techniques that were developed in prior work to calculate upper bounds on the worst-case cache related preemption delay (CRPD) of tasks [15]. For the calculation of WCCD, we use the worst-case number of hops between the source and target cores and the available network bandwidth as metrics. Further detail about the offline analysis of migration delay bounds is out of the scope of the current paper. Instead, we focus on the online policies to guide migration decisions.

4.2. Online Analysis: Choosing the Best Migration

In Section 3.2, we discussed several reasons that might trigger task migrations. In the current work, we focus on a subset of these triggers. Specifically, migration decisions are made a) when a periodic job is released and b) when a periodic job finishes execution. At the point where a migration decision needs to be made, offline migration delay bounds are first employed to determine whether a possible migration is feasible or not.

A migration is said to be *feasible* if the system remains schedulable in spite of the migration overhead introduced. In other words, a migration is feasible if and only if no task misses its deadline due to the migration. At a given time, there may be more than one feasible migration possible. In this section, we present techniques to choose a suitable migration candidate among a given set of feasible migration candidates. Since we only consider feasible migration candidates for comparison, safety of the system is guaranteed.

4.2.1. Comparing Feasible Migrations: Greedy Approach. Although migration delay bounds are necessary to determine the feasibility of a migration, they are pessimistic bounds due to the fact that they depend entirely on statically available information. In order to identify which of a set of feasible migrations is the least expensive, we present a set of online calculations that may be used to compare feasible migrations.

Migration overheads depend on several metrics. A weighted migration cost for a given migration candidate is calculated based on these metrics. Employing a *greedy* approach, the candidate with the lowest weighted cost is chosen for migration. It is to be noted that the weighted migration cost is a *relative* cost used to compare multiple feasible migrations. In the current work, we make a simplifying assumption that the target core of a migration is empty. In future work, we will consider the effects of a proposed migration on the tasks already allocated to the target core. The metrics considered in this work are described below.

1. Number of Cache Lines. When a task migrates, its cache lines have to be transferred to the intended target core. The worst-case number of cache lines that must be migrated, derived using offline MRCD values for the task under consideration, is a metric used in the calculation of the weighted migration cost.

2. Effect of Migration Mechanism. In Section 3.3, we briefly described the migration mechanisms developed in prior work [16]. The cost of migrating a given task to a specific target core depends on the underlying mechanism that facilitates the migration.

3. Time Until Next Release. As mentioned earlier, in the current work, we only migrate tasks at the end of a job so that the next job may start on the target core. Hence, the time available before the release of the next job of the migrated task is an important consideration while comparing multiple feasible migrations.

4. Distance to Target Core. This metric constitutes the worst-case number of hops between the source and target cores. In the current work, we assume that routes between cores are assigned statically, thereby simplifying the calculation of the number of hops. In future work, we intend to relax this assumption.

5. Quality of Service (QoS). The QoS factor of the network for communication among cores (NoC) is the minimum network bandwidth/latency that is guaranteed to be available along a route at a given point of time. The time taken for transfer of a set of cache lines along a given route is affected by this QoS factor.

4.2.2. Weighted Migration Cost. In this section, we present a method for calculating the weighted cost of a migration relative to other migrations. Equation 1 shows the calculation to determine the overhead of a migration with respect to the release time of the next job of the migration candidate. It is to be noted that migration time that overlaps with the time available before the release of the next job does not count as overhead as far as the response time of the migration candidate is concerned, although it affects the traffic on the NoC.

$$\hat{W}MC_i^{ts_m} = ((nc_i * m_i) * nh_s^t * q) - (t_i^{rel} - ts_m) \quad (1)$$

Here, i is the task number of the migration candidate T_i and ts_m is the start time of the potential migration. nc_i is the worst-case number of cache lines that need to be transferred between the source and target cores and nh_s^t is the worst-case number of hops between the source core (s) and the target core (t). m_i represents the effect of the migration mechanism used. q is the time required for the transfer of one cache line along one hop and represents the global QoS parameter for the NoC bandwidth and latency. t_i^{rel} is the release time of the next job of the migration candidate.

The effect of different migration mechanisms on the weighted migration cost is a comparative term among the mechanisms. It takes into consideration, the advantages and limitations of each mechanism. The less overhead a certain mechanism imposes on the transfer of cache lines between the source and target cores, the lower the value of the factor m_i. The subscript i is used here because the mechanisms affect different tasks in a different manner based on their cache access patterns.

As mentioned in Section 3.3, we assume migration of tasks uses one of three mechanisms, namely 1) Pull-based model (conventional warm-up, CW), 2) Whole Cache Migration (WCM) and 3) Regional Cache Migration (RCM). We also introduce the concept of an *Ideal Migration Mechanism* that assumes the migrated task's cache lines from the source cache are replicated on the target cache with zero overhead. In other words, it is as though there was no migration at all. It is to be noted that this ideal mechanism is not a realistic one, but rather serves as a point of reference for comparison. For ideal migration, we assume that $m_i = 0$ for all tasks. At the other end of the spectrum, we have conventional warm-up that imposes the maximum possible migration overhead since there is no explicit support for migration. For conventional warm-up, we assume that $m_i = 1$ for all tasks. For all other mechanisms, the value of m_i lies in between 0 and 1.

5. Conclusions

This paper presents greedy policies to guide migration decisions on a real-time multicore system. The policy considers the number of cache lines to be migrated, the mechanism being used for migration and characteristics of the NoC to choose the migration with the least overhead at any given time. It is expected that using such a policy to guide migrations will result in reduced response times for tasks and improved overall utilization of the system while guaranteeing real-time deadlines.

References

[1] J. Anderson, J. Calandrino, and U. Devi. Real-time scheduling on multicore platforms. In *IEEE Real-Time Embedded Technology and Applications Symposium*, pages 179–190, Apr. 2006.

[2] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Euromicro Conference on Real-Time Systems*, pages 35–43, June 2000.

[3] J. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In *Euromicro Conference on Real-Time Systems*, pages 76–85, June 2001.

[4] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *IEEE Real-Time Systems Symposium*, pages 119–128, 2007.

[5] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[6] A. Burchard, J. Liebeherr, Y. Oh, and S. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Trans. on Computers*, 44(12):1429–1442, 1995.

[7] J. Calandrino and J. Anderson. Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study. In *Euromicro Conference on Real-Time Systems*, pages 209–308, July 2008.

[8] D. Choffnes, M. Astley, and M. J. Ward. Migration policies for multi-core fair-share scheduling. *ACM SIGOPS Operating Systems Review*, 42:92–93, 2008.

[9] S. Dhall and C. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[10] D. Hardy and I. Puaut. Wcet analysis of multi-level non-inclusive set-associative instruction caches. In *IEEE Real-Time Systems Symposium*, Dec. 2008.

[11] T. Li, D. Baumberger, D. Koufaty, and S. Hahn. Efficient operating system scheduling for performance-asymmetric multicore architectures. In *ACM/IEEE Conference on Supercomputing*, Nov. 2007.

[12] T. Li, P. Brett, B. Hohlt, R. Knauerhase, S. McElderry, and S. Hahn. Operating system support for shared-isa asymmetric multi-core architectures. In *Workshop on the Interaction between Operating Systems and Computer Architecture*, June 2008.

[13] M. Moir and S. Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *IEEE Real-Time Systems Symposium*, pages 294–303, Dec. 1999.

[14] F. Mueller. Timing predictions for multi-level caches. In *ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems*, pages 29–36, June 1997.

[15] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemptions. *ACM Transactions on Embedded Computing Systems*, page (accepted), Mar. 2008.

[16] A. Sarkar, F. Mueller, H. Ramaprasad, and S. Mohan. Push-assisted migration of real-time tasks in multi-core processors. In *ACM SIGPLAN Conference on Language, Compiler, and Tool Support for Embedded Systems*, pages 80–89, June 2009.

[17] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *ACM Symposium on Theory of Computing*, pages 189–198, May 2002.

[18] J. Yan and W. Zhang. Time-predictable l2 caches for real-time multi-core processors. In *Work in Progress session of IEEE Real-Time Systems Symposium*, Dec. 2007.

[19] J. Yan and W. Zhang. Wcet analysis of multi-core processors. In *Work in Progress session of IEEE Real-Time Systems Symposium*, Dec. 2007.

[20] J. Yan and W. Zhang. Wcet analysis of multi-core processors with shared l2 instruction caches. In *IEEE Real-Time Embedded Technology and Applications Symposium*, Apr. 2008.