MemGuard: A Memory Bandwidth Management Framework for Real-Time Applications on Multicore Platforms

Heechul Yun (heechul.yun@ku.edu), Rodolfo Pellizzoni, Marco Caccamo, Lui Sha

Introduction

Multicore architecture is increasingly being adopted to many modern cyber-physical systems (CPS)—such as autonomous cars and unmanned aerial vehicles (UAVs)—that require high computing performance to process the vast amount of data flowing from a variety of sensors in real-time (e.g., obstacle detection and avoidance and real-time motion planning).

Challenges

Designing critical real-time applications on multicore architecture is, however, challenging because contention **in the shared memory resources** (e.g., memory bandwidth and cache space) can significantly alter the applications' timing characteristics. Recent trend toward heterogeneous multicore architecture---in which CPU and GPU cores share part of the memory controller---will likely cause even more contention because GPU tasks typically have **high memory bandwidth demands.** For example, memory intensive batch jobs running on CPU cores can cause significant delays to important real-time GPU tasks running in parallel, or vice-versa.

Unfortunately, today's real-time application developers have no good ways to address this problem. In traditional single core architecture, one can easily improve real-time performance by, for example, increasing the priorities of important real-time tasks. Raising task priorities, however, has no impact when tasks are running on different cores in parallel. This is a serious problem especially for safety-critical systems that need certification [1].

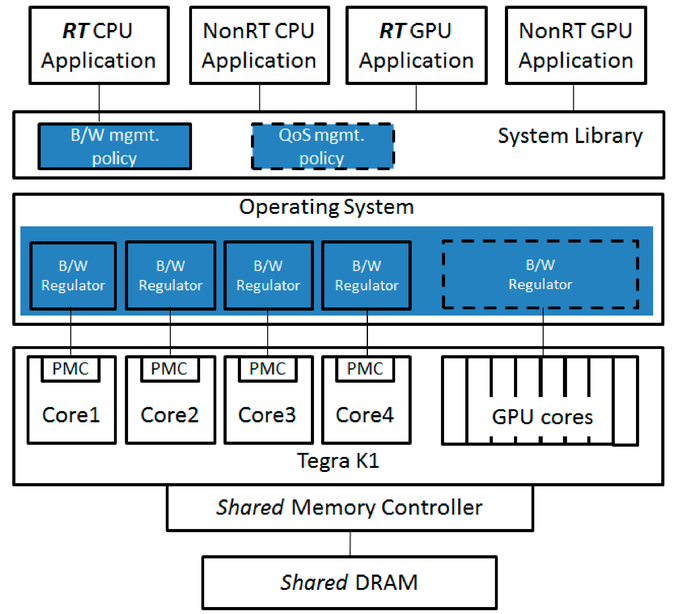
In this project, we present **a software framework to mitigate the memory contention problem** in heterogeneous multicore architecture.

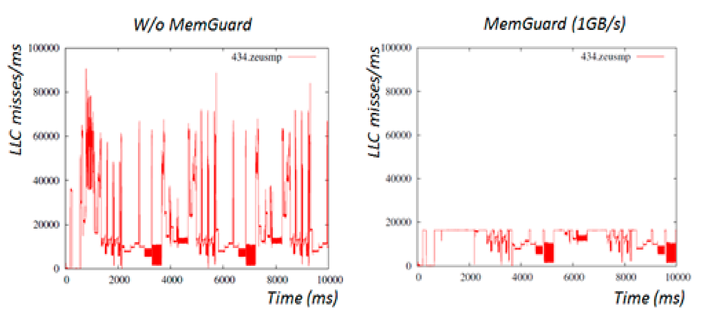
[1] Certification Authorities Software Team (CAST). Position Paper CAST-32: Multi-core Processors (Rev 0). Technical report, Federal Aviation Administration (FAA), May 2014.
[2] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2013
[3] H. Yun, S. Gondi, S. Biswas. Protecting Memory Performance Critical Sections in Soft Real-Time Applications, Technical Report, 2015
[4] https://github.com/heechul/memguard

The work is supported in part by NSF CNS 1302563.

The Framework

We developed a software framework, called **MemGuard**, to mitigate the memory bandwidth contention problem. The following figure shows the overall architecture of the system. The key idea is to periodically monitor and regulate the memory access rate of each core using per-core hardware performance counters (PMC) at the kernel scheduler. If, for example, a group of tasks generates too much memory traffic and delays the critical real-time tasks, MemGuard can regulate the memory access rates of the cores running the offending tasks.





Control Interface

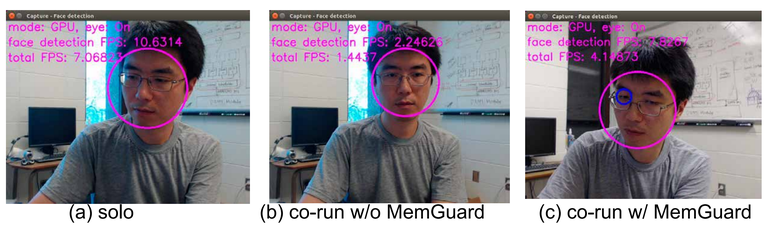
We currently provide core-level, task-level, and source-code level bandwidth control interfaces. The following shows an example of core-level bandwidth assignment. More details can be found in [4]

```
# echo mb 900 100 100 100 >
/sys/kernel/debug/memguard/limit
   → assign 900,100,100,100 MB/s for Core 0,1,2,3
```

Case study: real-time face detection in the presence of memory intensive co-runners

In this case study, our goal is to protect real-time performance of a face detection algorithm (from OpenCV package [3]) in the presence of memory intensive co-runners on the **Nvidia Tegra K1** multicore platform (4 ARM CPU cores + 192 GPU cores). The face detector is single threaded w.r.t. CPU but uses GPU cores to accelerate performance. We measured performance (frames/sec) of the face detector first alone in isolation—Figure 3(a); with three memory intensive co-runners—Figure 3(b); and with the co-runners that are bandwidth regulated using MemGuard—Figure 3(c). As shown in Figure 3(b), co-scheduling memory intensive co-runners significantly decreases the performance of the algorithm—from 8.6 to 1.8 fps on average. In contrast, Figure 3(c) shows that MemGuard significantly improves the performance—to 5.9 fps on average—by regulating the co-runners' memory access rates.

This results show that software-based memory bandwidth control can be effective in improving real-time performance of the face-detection software, which heavily utilizes GPU cores.



(a) solo (b) co-run w/o MemGuard (c) co-run w/ MemGuard

Face-detection performance comparison on Nvidia Tegra K1: (a) shows the performance of the face detection algorithm running alone on the system; (b) is when we launched three memory intensive co-runners; (c) is when we enabled MemGuard.

Ongoing Work

We are currently developing memory bandwidth management middleware that allow more fine-grained memory b/w control by the programmers. Our preliminary study shows that selectively applying bandwidth regulation to critical memory intensive code sections of real-time applications, which we call **memory performance critical sections**, can substantially improve real-time performance while minimizing impacts on the overall throughput. Our preliminary results on this extension may be available in [2].

Also, we are investigating mechanisms to regulate GPU's memory bandwidth usages. This is especially important for highly integrated platforms where CPU and GPU cores share memory subsystems---e.g., the Tegra TK1---as memory bandwidth becomes an even more serious bottleneck.