

Real-Time Computing on Multicore Processors

Lui Sha, Marco Caccamo, Renato Mancuso, Jung-Eun Kim, and Man-Ki Yoon,

University of Illinois at Urbana–Champaign

Rodolfo Pellizzoni, University of Waterloo

Heechul Yun, University of Kansas

Russell B. Kegley and Dennis R. Perlman, Lockheed Martin

Greg Arundale and Richard Bradford, Rockwell Collins

Although multicore technology has many benefits for real-time systems—among them, decreased weight and power, fewer cooling requirements, and increased CPU bandwidth per processor—multicore chips pose problems that stem from the cores interfering with one another when accessing shared resources. Interference is compounded in real-time systems, which are based on the assumption that worst-case execution time (WCET) is constant; that is, a software task's measured WCET must be the same whether that task executes alone or with other tasks. This assumption holds for single-core chips, but not for multicore chips unless they have isolation mechanisms between cores. Measurements we performed on a commercial multicore platform (Free-scale P4080) revealed that a task's WCET can increase by as much as 600 percent when a task on one core runs with logically independent tasks in other cores.

Because of the potential for large and random delay spikes, the US Federal Aviation Administration (FAA), European Aviation Safety Agency (EASA), and Transport Canada specify that only single-core chips can be used unless intercore interference is specifically defined and handled.¹ Indeed, *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, the primary document by which certification authorities such

Architects of multicore chips for avionics must define and bound intercore interference, which requires assuming a constant worst-case execution time for tasks executing on the chip. With the Single Core Equivalent technology package, engineers can treat each core as if it were a single-core chip.

as the FAA, the EASA, and Transport Canada approve all commercial software-based aerospace systems, was developed for the certification of software in single-core computers.² With a single-core chip, architects can assume a constant WCET and can thus schedule tasks and partition resources without unanticipated delays. Hence, the ideal solution is to certifiably bound intercore interference in a multicore chip such that each core can be used as a single-core computer.

As part of studying the feasibility of such a solution, we developed the Single-Core Equivalent (SCE) technology package, which addresses interference problems that arise when cores concurrently access DRAM, the memory bus, shared cache resources, I/O resources, and the on-chip network. With SCE, each core can be used as if it were a single-core chip, allowing the timing analysis and certification of software in a core independently of software in other cores. This has implications for avionics

RESEARCH FEATURE

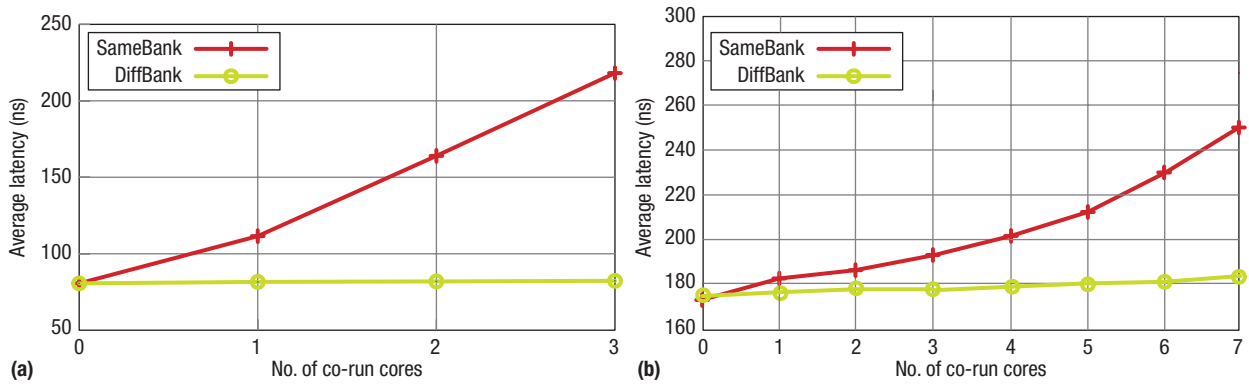


FIGURE 1. Effect of DRAM contention with a synthetic memory benchmark running on (a) the Intel Xeon and (b) the Freescale P4080 multicore chips. In the SameBank case, all cores access the same bank; in DiffBank, each core accesses a different private bank. Both graphs show that memory-access latency increases in the SameBank case as the number of cores running concurrently increases. The results imply that partitioning DRAM banks can reduce contention.

certification, as the D9178/B/C certification process targets avionics software in single-core computers.² With SCE, this process could work for multicore computers as well.

Our evaluations with the Freescale P4080, an eight-core chip, show that SCE successfully bounds intercore interference and removes unpredictable delay spikes.

MEMORY-RELATED INTERFERENCE

Memory-related interference is caused by conflicts in accessing memory and the memory bus. To resolve these sources of interference we created PALLOC, an OS-level memory allocator, and MemGuard, an OS-level memory-bandwidth manager, as part of the SCE package. Together, they increase performance isolation for applications that share DRAM.

Memory-access conflicts

DRAM is organized into ranks, banks, rows, and columns, with the greatest interference at the bank level. Figure 1 shows the average memory-access latency for a synthetic memory benchmark (linked-list traversal) when varying the number of interfering cores running the same benchmark. In SameBank, all cores access the same bank; in DiffBank, each core accesses a different private bank. In SameBank, memory-access latency increases as a function

of the number of concurrently accessing cores. However, in DiffBank, memory-access latency is not affected by other cores' activities. These graphs are evidence that performance isolation improves when each core has its own set of dedicated DRAM banks.³

Current OSs do not control how memory pages are mapped onto DRAM banks, which leads to poor performance isolation and unpredictable memory performance. Given a sufficient number of DRAM banks, PALLOC allows applications in different cores to access disjoint sets of specific banks.³

Memory-bus bandwidth

The synthetic memory benchmark associated with Figure 1 does not saturate the shared memory bus, so, when each contending core accesses a different bank, benchmark latency is barely affected. In contrast, multiple cores accessing memory-bus bandwidth can create a considerable bottleneck and thus increase interference. Because low-level memory arbitration policies in commercial hardware platforms are not known, we created MemGuard to manage memory bandwidth through a per-core bandwidth regulator for hard real-time applications.⁴

Per-core regulator. The per-core regulator monitors and enforces its corresponding allocation of core-memory bandwidth. Each regulator has a

memory-access budget Q_i for every regulation period P , which is global across cores. In an M -core chip, the sum of M memory-bandwidth reservations is equal to the system's sustainable memory bandwidth. When the given budget is exhausted, the regulator calls the OS scheduler to suspend computation on that core. At the beginning of each P , MemGuard replenishes the budget in full and the OS resumes the suspended tasks. P is a processor-wide parameter and shorter than the minimal application task period; currently, it is 1 ms. By restricting each core's maximum use of memory bandwidth, MemGuard effectively partitions memory bandwidth between cores and ensures strong performance isolation.

Global-bandwidth reclaiming. Bandwidth reservation alone could significantly waste available memory bandwidth because the core might not use all its reserved bandwidth and the reservable bandwidth that MemGuard can guarantee is much smaller than the hardware's peak bandwidth. To improve bandwidth use, MemGuard provides a bandwidth-reclaiming manager for soft real-time applications. At the beginning of the regulation period, the reclaiming manager estimates the cores' potential surplus bandwidth reservations and then redistributes on demand to the cores that need more bandwidth within the period. Available

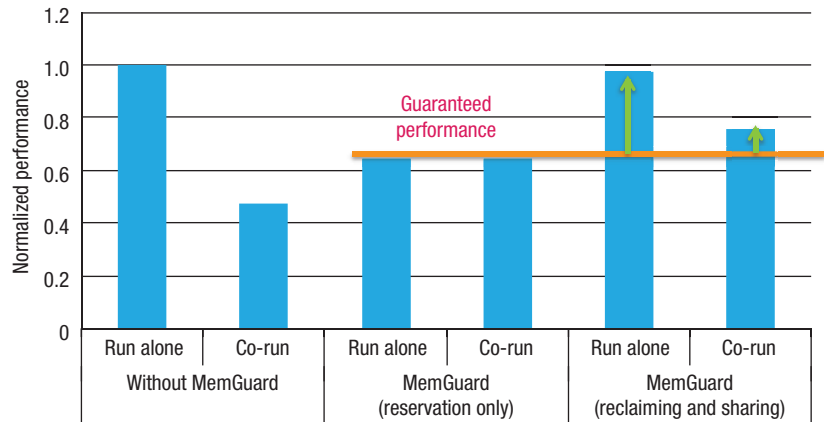


FIGURE 2. Performance impact of MemGuard. The y-axis shows the average instructions per cycle (IPC) for the 462.libquantum SPEC2006 benchmark when it runs alone (labeled run alone) and with memory-intensive co-runner (labeled co-run) in three different configurations: without MemGuard, MemGuard with only 1.0-GBps reservation, and MemGuard with both reservation and reclaiming.

bandwidth is greater than guaranteed bandwidth, so if the cores collectively exhaust the guaranteed bandwidth before the period ends, MemGuard lets them use the additional available bandwidth.⁴

Figure 2 shows an example of how MemGuard impacts performance. In this experiment, we measured the performance of the 462.libquantum SPEC2006 benchmark, first alone (labeled run alone) and then with a memory hog program (labeled co-run). Without MemGuard, the benchmark's performance dropped more than 50 percent when the benchmark was co-scheduled with memory hog. When MemGuard reserved memory bandwidth (1,000 MBps for libquantum and 200 MBps for the memory hog), performance of the libquantum benchmark decreased but was not affected by the memory hog. When MemGuard enabled reclaiming and then shared the reclaimed bandwidth among cores that needed it, performance improved in both the run-alone and co-run cases.

SHARED-CACHE INTERFERENCE

Modern CPUs feature at least one cache level organized as an associate set of a particular cache way. An associative set consists of cache lines with the same index. Depending on the running processes' addressing patterns, the cache controller loads data into the cache and writes data back from it in cache-line blocks. Each block can be loaded in any way and is chosen at fetch time according to the replacement policy. Once the cache way has been selected, the exact position inside the way depends on the value of a subset of the bits that compose the data address (index). Tag bits

are used to detect hits, while offset bits are used to address a particular byte in a cache block. SCE's cache-management approach avoids intercore interference through offline profiling and online allocation.⁵

Offline profiling

The memory-use profiler identifies the most frequently accessed virtual memory pages for a given executable. For each task, it produces a profile offline that ranks memory pages by access frequency. Because the address of a frequently used page is independent of its absolute virtual address, the profile is created only once, and virtual addresses are determined at runtime.

Online allocation

Online allocation consists of page coloring and lockdown of the last-level cache. Colored lockdown, a process that takes place after page coloring and lockdown, is the result of modifying the Linux kernel's page-management algorithm to make online allocation transparent to the application.⁶

Page coloring. Multiple DRAM pages mapped to the same set of shared cache pages have the same color and can be allocated across cache ways. Our OS techniques can reposition task memory

pages within the available colors to maximize allocation flexibility.

Lockdown. Because real-time applications are dominated by periodic execution flows with tight inner loops, it is possible to optimize use of the last-level cache by locking pages with the highest hit score first. In avionics applications that use the Integrated Modular Avionics (IMA) architecture, such pages can be preloaded in cache at the beginning of each IMA partition. The default configuration should evenly partition the last-level cache across cores.

Colored lockdown. Page coloring alone cannot guarantee that frequently accessed pages reside in cache, and, when lockdown is used alone, only a subset of frequently accessed pages can be allocated because, without coloring, there is no way to spread pages across multiple sets. Colored lockdown combines the two by first counting the number of frequently accessed pages with the same color. If, for a given color, the number of pages exceeds the number of available ways, the colored lockdown technique recolors extra pages into available sets and performs a lockdown on all the frequently accessed pages, including the recolored ones. With the combined approach, the total

RESEARCH FEATURE

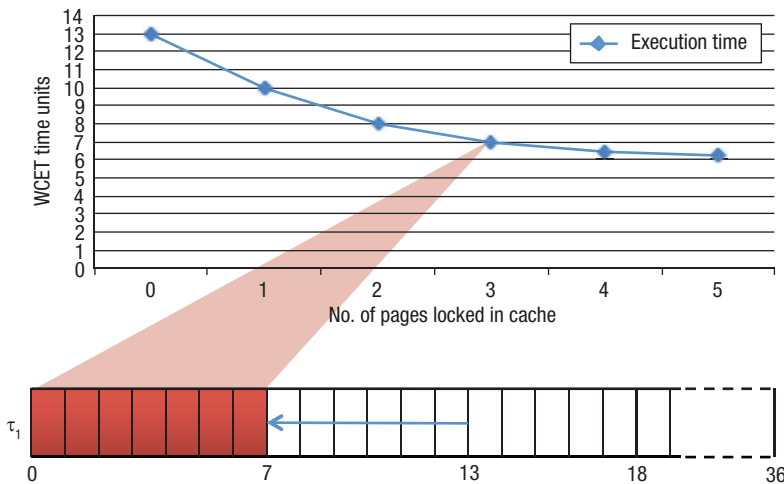


FIGURE 3. A lockdown curve of execution time when cache allocation is even, all but one core is idle, cores access private DRAM banks with PALLOC, and there is no MemGuard regulation. The task under analysis (τ_1) can use up to peak memory bandwidth to retrieve data. When three pages are locked, worst-case execution time (WCET) drops from 13 to 7 time units.

number of pages that can be locked in cache equals the size of the cache allocated to a core.²

By exploiting per-task data acquired through profiling, we fit a progressive lockdown PL curve that models a task's WCET as a function of the number of memory pages locked in the last-level cache. Figure 3 shows a sample PL curve.

INTEGRATED MODULAR AVIONICS

IMA is a common avionics architecture that uses time-division multiplex access (TDMA) to share a single-core computer's CPU cycles among applications, each of which uses a set of IMA time slots (partitions) in the IMA cycle.

SCE provides a solution to migrate applications residing in IMA partitions from single-core computers to fewer multicore computers. In IMA, there is one I/O partition for all application partitions' I/O activities, and IMA cycles in different single-core computers can have different lengths. Simply porting each IMA from a single-core computer to a core in a multicore computer is not the best approach because multiple I/O partitions in different cores could become active simultaneously, causing I/O conflicts in the shared I/O channel.⁷ To avoid this problem, SCE uses the I/O core—a dedicated core that

consolidates all I/O partitions. With the I/O core, IMA partitions in different cores can have different lengths for their major cycles, which makes I/O scheduling much easier. The I/O core has precedence relationships among physical and device I/O, shown in Figure 4.

As Figure 4 implies, I/O and processing partitions in the I/O core cannot overlap, but physical I/O transactions for different devices can be performed in parallel. To assign I/O and processing partitions, we developed the hierarchically ordered scheduling (HOS) heuristic algorithm.⁷ HOS starts by randomly but partially assigning the offsets of all physical I/Os and processing partitions and then finds a complete solution by determining the offsets of all device I/O partitions. Once the physical I/O and processing partition offsets are fixed, the search space for the device I/O partition offsets can be represented as a set of periodic intervals, which reduces problem size considerably. The HOS algorithm quickly finds a solution on average and scales well with problem size.⁷

SCE APPLICATION

The SCE application has two main stages. The first is to partition globally shared resources to create cores that are the equivalent of single cores. The

second is to estimate each task's WCET. Once WCET is obtained, each partitioned task's schedulability analysis is the same as for a single-core chip.⁸

Create single-core equivalence

Creating cores with single-core equivalence has five steps:

1. *Select the hardware.* The selected multicore chip must provide primitives that support last-level cache locking and the performance counters that MemGuard requires. For our experiments, we selected the P4080 chip.
2. *Ensure that each core has equal resources.* Each core should have private banks (PALLOC), an equal fraction of the memory bandwidth (MemGuard), and an equal amount of the last-level cache.
3. *Partition to allocate cores.* The allocation heuristics are well established for moving from slower single chips to a smaller number of faster single-core chips. We assume that the same heuristics apply in creating single-core equivalence.
4. *Allocate IMA partitions to cores.* Using the HOS heuristic, check if I/O channels can be partitioned temporally and meet all the precedence and capacity constraints. If not, return to the previous step. If there is no I/O solution, more or faster chips are needed. If there is an I/O solution, these four steps should yield a tentative set of virtual single-core chips.
5. *Optimize last-level cache partitioning and schedulability analysis.* Use colored lockdown to

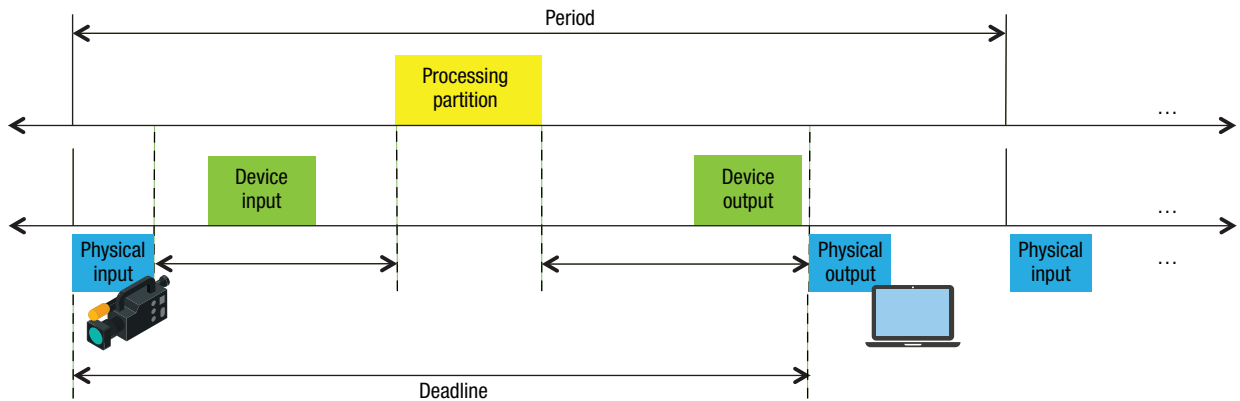


FIGURE 4. Partitions in the I/O core. The I/O core consolidates all I/O partitions by establishing precedence relationships among physical and device I/O. Each transaction is divided into physical and device I/O. Here, a camera taking pictures is the physical input. A device input transfers the buffered images to main memory, where an application within a core processes them. The device output—the processing result—is buffered in main memory and transferred to a physical output device; in this case, a monitor.

optimize the use of last-level cache partitions. For an M -core chip, in which $m \leq M$ cores will be used, estimate the $WCET(m)$ of each task and check the schedulability of partitions and I/O transactions. If the schedulability check fails, go to step 3. If there is no solution, more and/or faster chips are needed.

Estimate worst-case execution times

WCET is the foundation for schedulability analysis, which has been the focus of many analytic and experimental methods.⁹ SCE can reuse the WCET estimation methods developed for single-core chips and use the estimate to equally partition shared resources. Thus, if only one of eight cores is used, the core will have all the shared resources, and $WCET(1)$ will have the smallest WCET value. If applications never use more than two of the eight cores, it is possible to disable the remaining six cores and divide the total shared resources in half. For the same task, the shared resources then diminish with respect to the number of cores used, and WCET increases accordingly: $WCET(1) < WCET(2) \dots < WCET(8)$.

However, in a multicore chip, shared resources must be partitioned or this monotonic relationship might not hold because random intercore interference

can result in a $WCET(8)$ that is less than a $WCET(7)$. This random interference makes it impossible to estimate $WCET(8)$ with any certainty and complicates certification. Without first partitioning shared resources and then isolating the partitions, software running on one core could interfere with another core's software timing behaviors, making modular core-by-core certification impossible.

SCE addresses this problem by partitioning shared resources according to the cores being used. It first makes all but one core idle and estimates a task's $WCET(1)$ using traditional methods for a single-core chip. $WCET(1)$ is then used to calculate $WCET(m)$, where m represents the number of cores being used.

WCET(m) and schedulability. $WCET(M)$ represents the maximum intercore interference when all cores are used in an M -core chip. The more cores in use, the smaller the share of partitioned resources for each core, and the greater the overhead from partition-management software, which lowers schedulability for each core. If only $m < M$ cores are needed for the application now and in the foreseeable future, the remaining $(M - m)$ cores can be disabled and the shared resources partitioned into m chunks.

For that reason, SCE computes

$WCET(m)$ instead of $WCET(M)$. However, if an additional core is needed later on, SCE replaces $WCET(m)$ with $WCET(m+1)$ in each core's schedulability analysis.

Because $WCET(m+1)$ is bigger than $WCET(m)$, some applications in a core could become unschedulable and trigger the reallocation of applications to cores. In this case, recertification will likely be required, which is expensive. Hence, system engineers must carefully consider the ramifications of disabling cores in a multicore chip.

Estimating $WCET(m)$. The first step in estimating $WCET(m)$ is to look for last-level cache misses, which generate DRAM transactions. Let S_{line} be the cache-line size, which is measured as the number of bytes transferred during each transaction. S_{line} is architecture-specific and is provided in the chip specifications. Let L_{max} be the maximum delay on a DRAM transaction for the core under analysis. In analyzing worst-case delay, L_{max} is a key parameter and can be derived either experimentally or through DRAM analysis techniques.⁹

Experimentally, DRAM transfer bandwidth BW_{min} can be measured when two conditions hold: each memory transaction has a data dependency with the previous one and subsequent requests access different DRAM rows. Hence, L_{max} can be

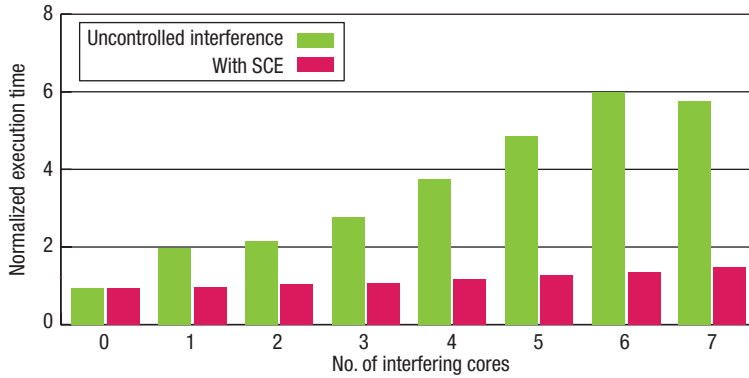


FIGURE 5. Effects of intercore interference for a single task running on the Freescale P4080 eight-core chip. The green bars are the worst-case finishing times (WCFTs) of a core 0 task when one to eight cores are used and intercore interference is uncontrolled. The red bars are the WCFTs of the same core 0 task when one to eight cores are used with the Single Core Equivalent (SCE) and a cache lockdown at 255 pages. WCFTs with SCE increase only slightly when more cores are used, because of increased isolation overhead and smaller size of shared resource partition. But the increase is much less dramatic, because random intercore interference through shared resources is removed. Data is from Lockheed Martin’s space systems testbed when porting single-core computers’ software to an eight-core Freescale P4080 computer.

derived from⁵

$$BW_{\min} = \frac{S_{\text{line}}}{L_{\max}} \tag{1}$$

We use Stall to designate the task delay induced by MemGuard regulation and compute it as⁵

$$\text{Stall} = \frac{m\mu S_{\text{line}}}{BW_{\min}} \tag{2}$$

where m is the number of active cores and μ is the maximum number of residual cache misses obtained from the PL curve. Each task’s WCET(m) includes the effect of bandwidth partitioning and is computed as Stall + C, where C is a task’s WCET(1), for the desired amount of allocated cache.

Equation 2 also provides useful insights into how DRAM bandwidth allocation impacts a task’s memory latency. BW_{\min} is clearly proportional to m and μ as well as to S_{line} . Additionally, stall time is inversely proportional to BW_{\min} . We performed a series of experiments that confirm Equation 2 provides a conservative estimate of a task’s WCET(m).²

An experimental way to estimate

the WCET(m) of a task in a particular core is to run tasks with worst-case interference in a nonstop loop in all cores.¹⁰ We recommend using both methods and checking if the experimental measure is always smaller than the theoretical estimation, which makes pessimistic assumptions to ensure it will be a valid bound.

PERFORMANCE EXAMPLES

We ran SCE technology in several experiments to validate its performance in bounding intercore interference. Figure 5 shows the reduction in delays of the same task running on P4080 with and without SCE.

When P4080 is used without SCE to control intercore interference, and seven of the eight cores were used, core 0 task’s worst-case finishing time (WCFT) increased 600 percent relative to its WCFT when only one core was used. In addition, random interference caused WCFT to peak when seven of the eight cores were used, not when all eight were used. This counterintuitive pattern makes it hard to determine the worst-case scenario needed to compute WCET(m) and complete certification.

Results from sample scenario

Figure 6 shows the system-wide scheduling solution from one of our more comprehensive experiments involving four tasks and three cores. WCET(3) denotes the use of only three cores, core 0 represents the I/O core, and cores 1 and 2 are the application cores. Following our SCE methodology, we first applied PALLOC to ensure that each core had a private set of DRAM banks. We then allocated portions of the last-level shared cache evenly for each core, partitioned memory-bus bandwidth evenly using MemGuard, and applied the HOS heuristic to find an I/O scheduling solution.

Figure 6 shows how different SCE techniques integrate. Colored lockdown is performed at the beginning of each partition instance. MemGuard and PALLOC are statically configured to evenly partition DRAM resources among cores. Finally, I/O operations are globally serialized over the I/O core. Each physical input precedes the corresponding I/O core input operation, and each physical output follows the corresponding I/O core output operation. Similarly, each I/O core input operation precedes the partition instance that consumes the corresponding data, and each I/O core output operation precedes the partition instance that produces the corresponding data.

For example, in core 1, IMA partition 1 has a period of 18 and a reservation of 6 time units, while IMA partition 2 has a period of 36 and a reservation of 12 time units. After colored lockdown, inside partition 1, task 3 has period 18 and WCET(3) 3; task 4 has period 36 and WCET(3) 3. Similarly, tasks 1 and 2 are running inside partition 2 with periods

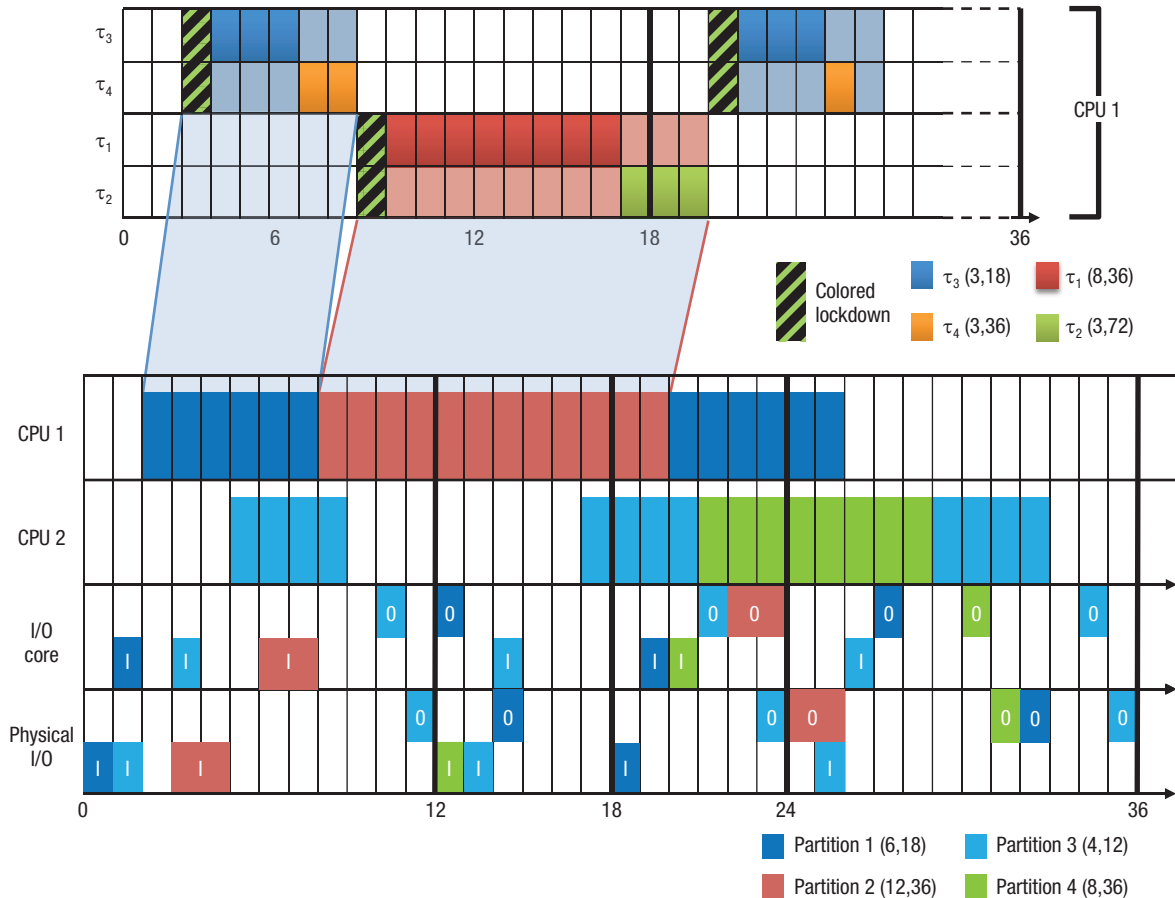



FIGURE 6. SCE solution for a three-core system running four tasks in which all SCE components are integrated.

36 and 72, respectively, and a WCET(3) of 8 and 3.

Making real-time computing efficient and predictable on multicore computers can greatly benefit many real-time applications, such as avionics, automotive, and medical-device control. It also enables avionics certification that has been problematic except when only using one core in a multicore chip. SCE provides a way to use established single-core estimation techniques to handle difficult schedulability analyses with multiple cores. We have already identified many areas for future work, including the certification of SCE itself.¹¹ An integrated solution to address memory consistency models, real synchronization protocols, and cache coherence protocols would benefit applications that must use multiple

cores in parallel. Other topics for additional research are how to incorporate fault-tolerance mechanisms and address backward compatibility with existing software applications. 

ACKNOWLEDGMENTS

The Single-Core Equivalent (SCE) technology was the result of a team effort. Sha led the architectural development and system integration. Pellizzoni and Zheng Pei Wu led the development of memory configuration and schedulability analysis. Caccamo and Mancuso led the development of last-level cache partitioning and hot code-segment lockdown. Yun and Gang Yao led the development of the mechanism to reserve memory bandwidth. Kim and Yoon led the development of the Integrated Modular Avionics (IMA) partition scheduling with conflict-free I/O for multicore systems. Kegley and Perlman guided the

development and transition from the perspective of Lockheed Martin, and Arundale and Bradford provided guidance from the perspective of Rockwell Collins.

This work was sponsored in part by National Science Foundation grants CNS 13-02563 and CNS 12-19064, by Natural Sciences and Engineering Research Council Discovery Grant 402369-2011, by Office of Naval Research grant ONR N00014-12-1-0046, by Lockheed Martin grant 2009-00524, and by Rockwell Collins's grant RPS 645038. Finally, we thank Nancy Talbert for her great assistance in the revision of this article. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors.

REFERENCES

1. Certification Authorities Software Team, "Position Paper CAST-32, Multicore Processors," rev. 0, 2014; www.faa.gov/aircraft/air_cert/design

- _approvals/air_software/cast/cast_papers/media/cast-32.pdf.
2. L. Sha et al., *Single-Core Equivalent Virtual Machines for Hard Real-Time Computing on Multicore Processors*, tech. report, CS Dept., Univ. of Illinois at Urbana–Champaign, 2014; www.ideals.illinois.edu/handle/2142/55672.
 3. H. Yun et al., “PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platform,” *Proc. IEEE 19th Real-Time Technology and Applications Symp. (RTAS 14)*, 2014, pp. 155–166.
 4. H. Yun et al., “MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multicore Platforms,” *Proc. IEEE 18th Real-Time Technology and Applications Symp. (RTAS 13)*, 2013, pp. 55–64.
 5. R. Mancuso et al., “WCET(m) Estimation in Multicore Systems Using Single Core Equivalence,” *Proc. 27th Euromicro Conf. Real-Time Systems (ECRTS 15)*, 2015, pp. 174–183.
 6. R. Mancuso et al., “Real-Time Cache Management Framework for Multicore Architectures,” *Proc. IEEE 18th Real-Time Technology and Applications Symp. (RTAS 13)*, 2013, pp. 45–54.
 7. J.-E. Kim et al., “Integrated Modular Avionics (IMA) Partition Scheduling with Conflict-Free I/O for Multicore Avionics Systems,” *Proc. IEEE 38th Computer Software and Applications Conf. (COMPSAC 14)*, 2014, pp. 321–331.
 8. L. Sha, “Real-Time Virtual Machines for Avionics Software Porting and Development,” *Real-Time and Embedded Computing Systems and Applications, LNCS 2968*, J. Chen and S. Hong, eds., Springer, 2004, pp. 123–135.
 9. H. Kim et al., “Bounding Memory Interference Delay in COTS-Based Multicore Systems,” *Proc. IEEE 19th Real-Time Technology and Applications Symp. (RTAS 14)*, 2014, pp. 145–154.
 10. R. Wilhelm et al., “The Worst-Case Execution Time Problem—Overview of Methods and Survey of Tools,” *ACM Trans. Programming Languages and Systems*, vol. 7, no. 3, 2008, article no. 36.
 11. L. Sha et al., “Position Paper on Minimal Multicore Avionics Certification Guidance,” 4 Jun. 2016; <https://1drv.ms/b/s!AqCnfGZqriHshuIGAczwiwEUa5ZjTQ>.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

Silver Bullet Security Podcast



In-depth interviews with security gurus.
Hosted by Gary McGraw.



www.computer.org/security/podcasts

*Also available at iTunes

Sponsored by




ABOUT THE AUTHORS

LUI SHA is the Donald B. Gillies Chair professor of computer science at the University of Illinois at Urbana–Champaign (UIUC). His research interests include software certifiability for real-time systems, medical best-practice guidance systems, and aeronautic technology for space programs such as the Mars Pathfinder and International Space Station. Sha received a PhD in computer science from Carnegie Mellon University. He is a Fellow of IEEE and ACM, a member of the NASA Advisory Council's Aeronautics Committee, and a co-recipient of the 2016 IEEE Simon Ramo Medal. Contact him at Irs@illinois.edu.

MARCO CACCAMO is a professor in the Department of Computer Science at UIUC. His research interests include real-time OSs, real-time scheduling and resource management, wireless real-time networks, and quality-of-service control in next-generation digital infrastructures. Caccamo received a PhD in computer engineering from Scuola Superiore Sant'Anna. He is a Senior Member of IEEE. Contact him at caccamom@gmail.com.

RENATO MANCUSO is a doctoral student in computer science at UIUC's Real-Time and Embedded Systems Laboratory. His research interests include OS-level techniques for embedded systems to enhance predictability, real-time-oriented development of heterogeneous platforms, and the deployment of unmanned aerial vehicles. Mancuso received an MS in computer engineering from the University of Rome Tor Vergata. He is a member of IEEE. Contact him at rntmancuso@gmail.com.

JUNG-EUN KIM is a doctoral student in computer science at UIUC. Her research interests include real-time scheduling; safety-critical, real-time multicore architecture; and cyber-physical systems. Kim received an MS in computer science and engineering from Seoul National University. Contact her at jekim314@illinois.edu.

MAN-KI YOON is a doctoral student in computer science at UIUC. His research interests include secure embedded systems, multicore architecture, real-time scheduling, and machine learning. Yoon received a BS in computer science and engineering from Seoul National University. Contact him at mkyoon@illinois.edu.

RODOLFO PELLIZZONI is an assistant professor in the Department of Electrical and Computer Engineering at the University of Waterloo. His research interests include embedded architectures, real-time OSs, and timing analysis. Pellizzoni received a PhD in computer science from UIUC. He is a member of IEEE. Contact him at rpellizz@uwaterloo.ca.

HEECHUL YUN is an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Kansas. His research interests include embedded real-time systems, OSs, and computer architecture. Yun received a PhD in computer science from UIUC. He is a member of IEEE. Contact him at heechul.yun@ku.edu.

RUSSELL B. KEGLEY is a Lockheed Martin Fellow at Lockheed Martin's Aeronautics Company. His research interests include schedulability analysis, advanced design techniques, middleware architecture, multiprocessor use in avionic systems, and cache-contention modeling and remediation for fighter aircraft programs. Kegley received an MS in computer science from Mississippi State University. He is a member of IEEE and ACM. Contact him at russell.b.kegley@lmco.com.

DENNIS R. PERLMAN is a senior research engineer at Lockheed Martin's Space Systems Company. His research interests include the design, development, and implementation of hardware-in-the-loop simulation testbeds for spacecraft programs. Perlman received an MS in applied mathematics from the University of Colorado. Contact him at dennis.r.perlman@lmco.com.

GREG ARUNDALE is a principal systems engineer at Rockwell Collins's Advanced Technology Center. His research interests include next-generation avionics architectures. Arundale received a BS in computer engineering from UIUC. Contact him at greg.arundale@rockwellcollins.com.

RICHARD BRADFORD is a principal systems engineer at Rockwell Collins's Commercial Systems division. His research interests include real-time scheduling, network modeling and simulation, optimization, and engineering economics. Bradford received a PhD in operations research from Stanford University. Contact him at richard.bradford@rockwellcollins.com.