Multicore Resource Isolation for Deterministic, Resilient and Secure Concurrent Execution of Safety-Critical Applications

Hamza Omar, Halit Dogan, Brian Kahne, and Omer Khan

Abstract—Multicores increasingly deploy spatial execution of safety-critical applications that demand a deterministic, resilient and secure environment to meet the safety standards. However, multicores aggressively share hardware resources that leads to non-deterministic performance due to destructive interference from concurrent applications. Resource sharing not only hinders efficient resilient execution, but also introduces security vulnerabilities due to information leakage on side-channels. This work proposes a novel multicore framework that constructs isolated clusters of cores for each concurrent application. It guarantees concurrent applications with deterministic performance, as well as an efficient execution environment for resiliency and security. Moreover, the framework allows dynamic re-sizing of cluster sizes for load balanced execution of concurrent applications. However, it leads to diminished isolation between clusters, which opens various performance–resilience and performance–security tradeoffs.

Index Terms—Multicore, hardware resource sharing, safety-critical systems, resilience, security, side-channels.

1 Introduction

Multicore processors are thriving, and much attention is devoted to building resource sharing schemes that allow concurrent applications to utilize on-chip hardware resources. These machines offer immense computation capabilities, making them attractive for deployment in numerous safety-critical environments [13], [18]. However, concurrent execution of applications promote space-time sharing of hardware resources, such as on-chip network, private and shared caches, and off-chip memory. This resource sharing induces interference channels that lead to undesirable performance effects in systems executing concurrent applications. It has been shown that resource isolation limits interference as each application utilizes its dedicated hardware resources [5], [9]. Building on the idea of resource isolation, a novel framework is proposed that partitions multicore resources to form isolated clusters of cores. By reverse engineering, a prototype of the proposed isolation framework is developed on the *Tilera® TileGx72™* multicore processor. To the best of our knowledge, no prior work has been done in the context of incorporating isolation principles to build an execution environment for efficient, resilient, and secure execution of concurrent applications.

Interference channels between concurrent applications incur non-deterministic timing behaviors in multicores. In this paper, resource isolation at the granularity of clusters of cores is shown to demonstrate deterministic performance guarantees. Furthermore, real-time systems are interconnected that require secure interactions between both cyber and physical worlds [3]. Recent works have shown that an adversary can gain access to critical information through side channels in the shared execution hardware [20]. Moreover, one compromised application in a concurrent execution environment can cause others to perform undesirable actions, or even take control of the entire system [16]. Therefore, it is of utmost importance to provide safety-critical applications with a safe and secure execution environment. As isolation is considered essential for security [7], [17], the proposed framework limits information leakage via resource isolation.

Real-time systems often operate in harsh settings, such as high radiation environments. Multicores introduce numerous challenges for protection against transient faults due to their complex communication and memory access protocols that aggressively share on-chip resources. Indeed, it is critical to provide resiliency guarantees for safety-critical applications deployed on multicores. Among various resilience schemes [6], [11], recent work from AMD [19] provides high soft-error coverage by adopting *dual-modular redundancy* (DMR) to execute two copies of the same application. However, redundant execution incurs significant performance degradation over a system with no redundancy [19]. This degradation is attributed to loss of thread-level parallelism, as well as destructive interference effects due to aggressive resource sharing. This paper makes a key insight that isolating shared resources leads to improved performance for the DMR scheme executing two copies of the same application. The proposed isolation framework creates isolated clusters where redundant execution does not incur destructive interference effects. Consequently, a performance improvement of ∼25% is observed for iterative decision algorithms over DMR with no resource isolation. Moreover, the proposed DMR setup provides stringent resiliency guarantees, as it allows an application to continue execution even when the other has malfunctioned or crashed.

The proposed framework also realizes a *performance-adaptive* method that dynamically varies the cluster sizing to create an efficient yet deterministic execution environment at the granularity of concurrent application instances. It opens up new optimizations for *performance–resilience* and *performance–security* tradeoffs. For resilience, a novel selective approach is proposed that allows a certain percentage of the application iterations to execute in dual modular redundancy mode, while the remaining iterations execute in a single cluster to exploit all available multicore parallelism. This allows the framework to dynamically vary cluster sizing to exploit performance benefits, however, it leads to lower output accuracy and error coverage (resilience). Furthermore, *performance-security* tradeoff is explored where cores are added/removed via dynamic clustering to achieve better performance through higher thread-level parallelism. However, it exposes certain side-channels for an adversary to infer confidential information, leading to diminished isolation. In a nutshell, the proposed isolation framework is shown to provide safety-critical systems with a performance-adaptive execution model that effectively trades off security and resilience for leakage-adaptive execution, alongside protection against soft-errors to avoid system failures.

2 Multicore Isolation Framework

The proposed isolation framework is prototyped on *Tilera® TileGx72™* processor, which is a tiled multicore architecture consisting of 72 tiles, connected using an intelligent 2-D mesh on-chip network. Each tile consists of a 64-bit VLIW core, private level-1 data and

- *H. Omar, H. Dogan, and O. Khan are with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, 06269. E-mail: {hamza.omar, halit.dogan, omer.khan}@uconn.edu.*
- *B. Kahne is with Automotive Microcontrollers and Processors, NXP Semiconductors Inc., Austin, TX, 78735. E-mail: brian.kahne@nxp.com.*

Manuscript recieved 17-May-2018, revised 8-Aug-2018, accepted 9-Sep-2018.

instruction caches, and shared level-2 (L2) cache. A directory is integrated at the L2 cache for directory-based cache coherence protocol. Additionally, the total capacity across each core's L2 cache is viewed as a large level-3 cache. It offers various configurations for data placement and caching schemes. The default placement utilizes *hash-for-home* scheme that interleaves cache lines across L2 slices. By default, data is also interleaved among all 4 on-chip memory controllers. *Tile-Gx72™* allows execution of multiple concurrent applications using `fork()` and `exec()` system calls. Moreover, it includes a switching engine to communicate between neighboring tiles, I/Os and the on-chip memory controllers. It also provides support for core-to-core direct messaging using special User Dynamic Network (UDN), where threads are pinned to cores.

2.1 Establishing Isolation on Tilera® Tile-Gx72™

Using the re-configuration capabilities of *Tile-Gx72™* processor, an isolated environment is created for interference-free execution of multiple concurrent applications. The number of isolated clusters that can be formed is a function of available on-chip memory controllers, and thus, four completely isolated clusters can be constructed using *Tile-Gx72™*. However, for ease of explanation, two clusters are considered.

2.1.1 Clustering & Isolating the On-Chip Network

In *Tile-Gx72™*, application threads are spatially distributed among available cores. To form clusters of core(s), each cluster (say CL_1) is assigned with a set of cores (say CPU_1) to execute an application instance APP_1. In order to allow CL_1 to execute APP_1, threads from the application are pinned to the cores assigned to that cluster, using *Tilera's®* API call `tmc_cpus_set_my_cpu (tid)`. Similarly, the other cluster (CL_2) is formed by assigning the respective cores (CPU_2) for executing APP_2. Note that the cores assigned to both clusters should never overlap each other, i.e., $CPU_1 \cap CPU_2$ must be $\emptyset$, otherwise applications experience interference among each other. Additionally, for each cluster, the network traffic is routed such that all requests and data packets remain within the boundary of the cluster. *Tilera® Tile-Gx72™* implements a smart *X-Y routing* algorithm with 2-D mesh network topology. With *X-Y routing* in place, the proposed framework isolates all available networks – including UDN – by routing each packet to/from the allocated memory resources. In the next subsections both on-chip and off-chip memory resources are allocated to clusters so that no packet would drift towards other clusters' domain. Figure 1:Step ① shows the formed clusters where CL_1 is provided with first 32 cores of the processor, while last 32 cores are allocated to the cluster CL_2.

2.1.2 Isolating On-Chip Memory

Tile-Gx72™ offers various configurations for caching and data placement. Default placement utilizes *hash-for-home* scheme, where an entire page is hashed across all tiles (L2s) at the cache line granularity. However, hashing data among all tiles violates isolation as one cluster's data may be mapped outside the cluster boundary – leading to interference among concurrent applications. Therefore, it is important to keep one cluster's data within its own set of cores to avoid interference. To limit resource sharing, *local homing* is utilized since it allows the programmer to *pin* pages/data to any specific L2. This is done using *Tilera's®* API call `tmc_alloc_set_home`[1] `(&allocation_type, core_id)`. Moreover, *L2-replication* is disabled with *local homing* to ensure that each L2 slice is accessed by a single application. Figure 1:Step ② shows specified cores for each cluster to map their respective data.

[1] *Tilera® Tile-Gx72™* specific *TMC_ALLOC_INIT* is used as the `allocation_type`, and `core_id` specifies the core/L2 to map the page/data.

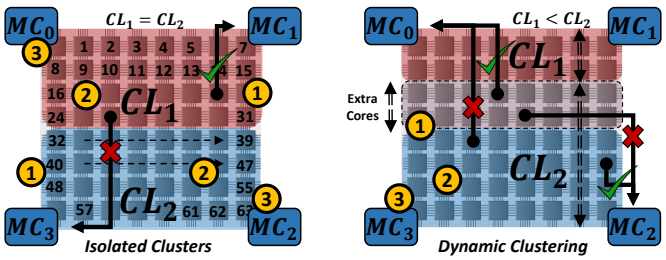

Fig. 1: Proposed framework with isolated clusters. The flexibility to dynamically vary cluster sizes is also shown.

2.1.3 Isolating Off-Chip memory

To ensure chip level isolation, all means of data accesses must be isolated among clusters. Figure 1 shows the memory controller IDs for *Tile-Gx72™*, where each on-chip memory controller is attached to an independent physical memory channel. Therefore, routing each clusters' L2 misses and data requests to a memory controller is sufficient to achieve off-chip memory isolation. This allows each cluster to access an independent physical channel, a memory bank, and a memory row. The cluster CL_1's access to off-chip DDR memory components is realized by forwarding its traffic to the MC_0 and MC_1. This is done by *Tilera's®* API call `tmc_alloc_set_nodes_interleaved(&allocation_type, pos)`, where `pos` is set to the bit mask of respective memory controllers to be activated. For example, CL_1 will have `pos = 0b0011`. Similarly, `pos` for CL_2 is set to `0b1100` – access to memory controllers MC_2, and MC_3 (c.f. Figure 1 – Step ③). This step guarantees no off-chip interference among the two clusters of cores.

3 Safe/Resilient Concurrent Execution Environment

The proposed cluster-level isolation framework ensures no sharing across on-chip hardware resources, and thereby allows interference-free deterministic execution. Prior works [4], [12] have also focused on scheduling and efficiently utilizing resources based on the expected impact of interference in a multicore setting. In similar context, the proposed framework enables a capability to dynamically vary the number of cores assigned to each cluster in such a way that only interference in the on-chip network and private caches is allowed for the cores that are re-allocated to execute threads from the other cluster. This runtime *Dynamic Clustering* is shown on the right image of Figure 1, where CL_2 uses two rows of cores to map additional threads, while CL_1 still has access to the L2 caches of these cores. After dynamic re-configuration, CL_2 is provided with 16 more cores ($16-63$) for increased thread-level performance, while its data/memory resources are unchanged, i.e., L2s from cores $32-63$, and memory controllers MC_2 and MC_3. For CL_1, it now has lower thread-level parallelism (only cores $0-15$ for its threads) but still has access to its initially allocated data/memory access resources, i.e., L2s from cores $0-31$, and memory controllers MC_0 and MC_1. Using this capability, one can allocate a larger/smaller cluster size to a resource hungry application for efficient execution. A cluster is allowed to only steal core resources from the other cluster, while the L2 and memory controller resources are not altered during runtime. This is done to avoid performance overheads from data/memory re-allocations that are performed once at each application startup. Therefore, *Dynamic Clustering* is expected to underperform slightly as compared to a static re-configuration that optimally re-assigns on-chip resources to each cluster. Using this feature, a *performance-adaptive* capability is built in the proposed framework that dynamically takes away some cores from a cluster, and assigns them to another cluster in need of more computational power. This allows concurrent applications to load balance their performance needs while still guaranteeing a certain level of determinism.

3.1 Leakage-Adaptive Secure Execution

The proposed isolation framework brings about a *performance-security* tradeoff space via dynamic clustering. Referring to the right image of Figure 1, when clusters (CL_1 and CL_2) are dynamically re-configured by adding cores to CL_2, the *extra cores* become the source of interaction between two clusters and lead to diminished isolation. Each cluster is still isolated for most of its data/memory resources, but the on-chip network resources may still observe interference by CL_1 due to messages routing across the cluster boundaries to access its L2 slices. Moreover, the data of both clusters temporally co-locate in the private L1 caches of the *extra cores*, creating an L1 cache interference channel between applications. Such cases can potentially lead to various side-channel exploits, where an adversary can infer secret knowledge by monitoring these interference channels.

The side-channel threats can be dealt with by invalidating the entire L1 cache of the *extra cores* during dynamic re-configuration of clusters. *Tilera® Tile-Gx72™* processor features various caching modes, such as enabling/disabling L1 or L2-replication. Similar level of security can be achieved by disabling cache replication and allowing these cores to only perform a regulated *word-by-word* remote L2 access to CL_2. Both the solutions presented above will allow segregation among the data of CL_1 and CL_2 in *extra cores*. The former solution allows L1 cache to hold one cluster's data at a time, whereas the latter solution allows the *extra cores* to either not use the L1 cache at all or only allow one cluster to make use of it. Regardless, these solutions incur performance overheads due to expensive cache flush/invalidate operations, or word-level remote accesses that may not fully exploit data locality in the L1 caches. Therefore, with such a tradeoff it is natural to ask: *how much information leakage would be acceptable to achieve efficient execution, along with an adequate level of security?* Investigating this aspect of the proposed framework will be a topic of future work.

3.2 Resilient Execution Environment

The proposed isolation framework can be employed to ensure a resilient execution environment. For example, *n-modular redundancy* (nMR) can be implemented on *Tilera® Tile-Gx72™* processor, where n instances of the same application are concurrently executed and checked for output correctness. For illustration of this capability, the proposed framework is utilized to demonstrate a *dual-modular redundant* (DMR) execution environment. It concurrently executes two copies (instances) of the same application (say *APP*). To limit the adverse effects of interference, both instances are executed on isolated clusters, i.e., instance A_{APP} is executed on CL_1, while instance B_{APP} on CL_2. Upon completion of A_{APP} and B_{APP}, results from both instances are verified to ensure correct execution. This is done using *Tilera®*'s user dynamic network (UDN). Each cluster computes a *32-bit XOR hash* of the application output updates, and sends the hash over to the other cluster via UDN. The received hash is compared against the locally computed hash value. A roll-back mechanism is developed to re-execute DMR processes if *hash values* from both instances do not match. Otherwise, the applications are allowed to terminate successfully.

3.2.1 Selective Resilience for Efficient Execution

The isolation framework mitigates performance degradation from interference of shared resources due to DMR execution. However, resiliency leads to loss of parallelism, and hence performance. For safety critical systems, both resiliency and performance vary based on the conditions and constraints surrounding the system. Hence, it is beneficial to selectively incorporate resiliency for an application, such that performance and error coverage demands are both fulfilled simultaneously. Prior works [8], [15] have shown to improve performance by employing resiliency selectively. Moreover, many analytics

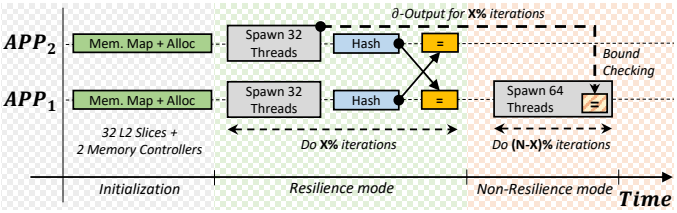

Fig. 2: Design flow of selective resilience scheme for the proposed framework. APP_1 runs non-crucial iterations in *non-resilience* mode. APP_2 sits idle, and waits until the *resilience* mode is initiated.

and iterative applications have been shown to benefit from selective resilience by partitioning the application into crucial and non-crucial regions [11], [14]. For crucial code regions, DMR execution can be used to ensure high resiliency coverage, while non-crucial regions execute efficiently by exploiting full thread-level parallelism via cluster re-configuration capabilities described earlier (c.f. Section 3).

As shown in Figure 2, an efficient selective resiliency mechanism is proposed for iterative algorithms that utilize the dynamic cluster re-sizing capability to guarantee both resiliency and efficiency. The scheme initializes two clusters, CL_1 and CL_2, with half of the cores, L2s, and memory controllers allocated to each cluster. Initially, for resilience mode, both APP_1 and APP_2 are executed using 32 threads each for a certain number of iterations (say **X**%), and compute an intermediate output (*∂-Output*). To ensure correct resilient execution, *∂-Output* verification is done for both clusters by comparing the 32-bit hash of the outputs (c.f. Section 3.2). Upon completion of resilience mode, the system is switched to *non-resilience* mode to execute remaining **N-X**% iterations, where CL_1 is provided with 64 cores to exploit thread-level parallelism. However, CL_1 still executes APP_1 with half of the L2 and memory controller resources. To achieve output correctness, a *bound checking* process is added to verify that the computed results are within certain bounds. These bound checks are *fine granular*, as each thread is enforced to check the result it produces before it commits to the output data structures. Upon failure, the bound checking step uses *∂-Output* (an already verified intermediate output) as a back up, and either commits it to the final output or deploys some other error correction strategy. The *bound checker* checks for lower and upper cut-off values which vary from one application to the other. For applications which have predetermined bound values, these values can be directly employed for bound checking purposes. Otherwise, the bound values are approximated using the intermediate *∂-Output*.

4 Methodology

The *Tile-Gx72™* multicore implements 72 tiles with each tile featuring a 64-bit VLIW core, 32 KB L1-I/D caches, and a 256 KB shared L2 cache slice. The off-chip DRAM memory is accessible using four on-chip 72-bit ECC protected DDR3 controllers that are attached to independent physical memory channels. Moreover, it consists of 5 independent 2-D mesh networks with *X-Y routing*, one for on-chip cache coherence traffic, one for memory controller traffic, and others for core-to-core and I/O traffic. To isolate shared hardware resources, the default *hash-for-homing* scheme is overridden to use the *local homing* scheme for pinning data structures on specified L2s. Overheads introduced by the use of *local homing*, data checker mechanism, and switching from *resilience* to *non-resilience* modes are all considered in the completion times of the applications.

All simulations are conducted by utilizing 64 of the 72 available tiles. Two graph benchmarks, Single Source Shortest Path (SSSP) and PageRank (PR) are acquired from the Pannotia [2] and CRONO [1] benchmark suites, respectively. These graph workloads are executed using a real world California Road Network graph [10]. Moreover, a

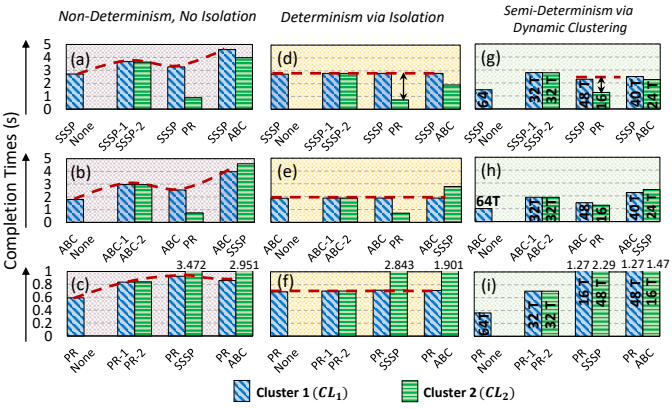

Fig. 3: Impact of resource sharing on deterministic execution of concurrent applications with/without isolation.

mission planning algorithm, Artificial Bee Colony (ABC) is adopted from advanced driver-assistance system (ADAS).

5 Evaluation

5.1 Deterministic & Secure Execution via Isolation

Figure 3 shows the impact of resource sharing on the performance of concurrent applications, where each cluster is provided with 32 cores. For each experiment ((a) – (i)), cluster CL_1 always runs a fixed reference application, and CL_2 is provided with different application scenarios, such as None refers to no application. For any workload x, $x-1$ and $x-2$ identifies two instances of x executed on CL_1 and CL_2 respectively. Figures 3 – (a) to (c) refer to the concurrent execution with default parameters, and *no isolation*. Therefore, the reference application when executed with None exhibits the ideal completion time. However, when a co-located application induces interference due to resource sharing, the completion times for the reference application vary non-deterministically. When the applications are executed under the proposed isolation setup (Figures (d) – (f)), the completion times for the reference applications stay constant and thus predictable. In the context of secure execution, it has been theoretically shown that isolation of shared resources serve as a fundamental primitive for a secure execution environment [17]. Therefore, no interference across concurrent applications in Figures (d) – (f) enables side channel attack mitigation. In such an environment, even if one cluster gets compromised, an attacker would require ample amount of resources to infer any knowledge from another concurrent safety-critical application executing on the other cluster.

Clearly, in some cases there is imbalance in terms of how concurrent applications complete. Consider the SSSP–PR tuple in Figure 3:(d). PR completes ∼3.5 times faster as compared to SSSP, when both applications have equal resources. It would be helpful to rebalance resources to tackle the variation is utilization. As discussed in Section 3, cluster sizes can be varied in a performance adaptive fashion. Figure 3:(g) shows that when SSSP is provided with 48 cores/threads and PR with 16 cores/threads, the proposed framework exhibits more balanced completion numbers (difference reduced to ∼1.5× from ∼3.5×). However, by doing so the proposed scheme only guarantees deterministic execution at the *application–core-count* tuple granularity. For example, Figure 3:(i) shows that PR at 16 cores/threads demonstrates deterministic completion time when coupled with SSSP or ABC. The idea of dynamic clustering leads to load balanced execution, while ensuring a semi-deterministic execution environment. However, it also introduces a *performance–security* tradeoff, as adding or removing resources among clusters introduce certain interference channels that potentially leak information. This aspect of the proposed framework will be a topic of future research. However, the application of the proposed framework for system resiliency is discussed in detail next.

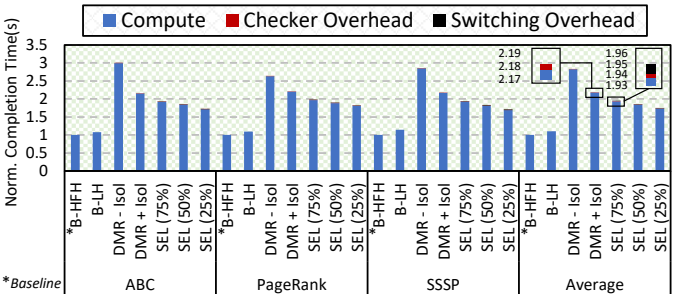

Fig. 4: Normalized completion times for all workloads with DMR ± Isol and SEL ± Isol. Results are normalized to the baseline utilizing *hash-for-home* i.e., B–HFH.

5.2 Isolation-based Resilient Execution

Figure 4 shows the normalized completion times for all workloads when executed under dual-modular redundancy, with and without isolation mechanisms (DMR + Isol, and DMR − Isol respectively). Results are normalized to the baseline (B-HFH) which runs the application with 64 cores, *hash-for-homing* scheme and all available memory controllers. B-LH is considered to show the overheads incurred by the use of *local homing* scheme. On average, *local homing* incurs an overhead of ∼10.5% over *hash-for-home*.

The "DMR − Isol" scheme incurs an average overhead of ∼2.84× over the baseline, primarily due to loss of thread-level parallelism and interference on shared hardware resources. In the context of interference, the application instances compete for same shared cache resources that lead to increased stress on the available cache capacity and reduced data locality. Moreover, increased L2 misses stress the memory controller queues leading to higher contention delays to access main memory. Finally, application instances generate traffic that interferes in the routers, causing high contention delays in the on-chip networks. The "DMR + Isol" scheme elegantly isolates the resources among the clusters to limit interference, and reduces DMR performance overheads from ∼2.84× to ∼2.18× – resulting in an improvement of 24%. The reported completion times in Figure 4 also include the UDN *checker* overheads. These overheads include the time taken by both clusters to compute, send, receive, and compare the *32-bit XOR hash*. On average, the UDN *checker* incurs an insignificant overhead of less than 1%.

5.2.1 Selective Resiliency

The selective resilience scheme discussed in Section 3.2.1 is evaluated for the three iterative optimization benchmarks. The application software requires the necessary bound checking mechanisms during the non-resilient iterations of each benchmark. For SSSP, the upper bound is set at the ∂-*Output* values of the *distance array* since each vertex relaxation step is always guaranteed to decrease as iteration counts increase. If any *distance array* value increases in non-resilient mode, it indicates a soft-error perturbation. However, determining the lower bound for each vertex *distance array* value is challenging since the shortest distance update calculation is data dependent. The strategy to determine the lower bound relies on approximating its value to decrease no more than a pre-determined percentage of the ∂-*Output*. This percentage value is application and system dependent and can be set in the software. For the purpose of evaluation, this work employs a ∂-*Output* percentage cut-off of 15%. For PR benchmark, the *rank values* for each vertex are predetermined and must stay in the range of 0 and 1. Similarly, for ABC, the output vector (*acceleration, velocity, jerk*) values also have predetermined lower and upper bound values. Therefore, both PR and ABC utilize their predetermined bounds for their checker mechanisms during the non-resilient mode of execution.

Figure 4 shows the average normalized completion time breakdowns for the selective resilience switching scheme (SEL). Selective resilience executes **X%** of iterations (as suggested by SEL (X%)) in the *resilience* mode. The *resilience* mode operates with two clusters whose data structures are pinned to their respective cores. When switched to the *non-resilience* mode using dynamic clustering, one cluster is provided with all available resources i.e., 64 cores/threads, where it executes the remaining iterations to achieve better performance. However, the active cluster still uses the same resources defined during the initialization of the framework, i.e., 32 L2 slices, and 2 memory controllers. The *switching overhead* includes (1) the overheads for switching from *resilience* to *non-resilience* mode, and (2) the overheads for performing the *bound check* comparisons. Compared to DMR + Isol, selective resilience shows a performance improvement of 20% for SEL (25%) configuration that executes the first 25% of application iterations in resilience mode, and the remaining in non-resilience mode. Overall, the proposed resilience schemes improve performance over "DMR − Isol" by (1) creating isolated clusters of cores for redundant execution of application instances, and (2) exploit the *performance–resilience* tradeoffs using selective resilience.

6 Conclusion and Future Work

This paper introduces a novel multicore framework that leverages isolation at the granularity of clusters of cores to provide deterministic execution of multiple concurrent applications, along with efficient resiliency and security guarantees. The preliminary evaluation shows that deterministic performance can be guaranteed even when the cluster sizing is dynamically varied to load balance application performance. Moreover, dual-modular redundancy is shown to work more efficiently under isolation of shared resources as compared to traditional setup where shared resources are allowed to interfere among the redundantly executing application instances.

In future, we plan to study the scalability of static and dynamic clustering schemes for futuristic multicores with higher core counts. Furthermore, we plan to characterize the benefits from removing the hardware interference channels among concurrently executing applications using micro-benchmarks and a wide range of application domains. We will also explore the various *performance–resilience* and *performance–security* tradeoffs. For resiliency, we plan to conduct studies to evaluate the impact of selective resilience scheme on the error-coverage and output accuracy of the deployed applications. In terms of security, we plan to further study various side-channel exploits introduced by dynamic clustering, and conduct a security analysis by performing various attacks to show the security of the proposed framework. We plan to quantify information leakage for the target applications, and employ various architectural mechanisms for mitigation purposes, leading to no or managed information leakage between concurrently executing applications.

Acknowledgments

This research was partially supported by the National Science Foundation under Grants No. CCF-1550470 and CNS-1718481.

References

[1] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan. Crono: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores. In *2015 IEEE IISWC*.

[2] Shuai Che, Bradford M. Beckmann, Steven K. Reinhardt, and Kevin Skadron. Pannotia: Understanding irregular gpgpu graph applications. *2013 IISWC*, pages 185–195, 2013.

[3] Chien-Ying Chen, Monowar Hasan, and Sibin Mohan. Securing real-time internet-of-things. *CoRR*, abs/1705.08489, 2017.

[4] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *ASPLOS*, pages 127–144, 2014.

[5] D. Elkaduwe, P. Derrin, and K. Elphinstone. Kernel design for isolation and assurance of physical memory. In *IIES' 08 Workshop*.

[6] S. Feng, S. Gupta, A. Ansari, and S. Mahlke. Shoestring: Probabilistic soft error reliability on the cheap. *SIGPLAN 2010*, pages 385–396.

[7] S. K. Haider, H. Omar, I. Lebedev, S. Devadas, and M. van Dijk. Leveraging hardware isolation for process level access control & authentication. In *2017 ACM SACMAT*.

[8] D. S. Khudia and S. Mahlke. Harnessing soft computations for low-budget fault tolerance. In *MICRO*, pages 319–330, Dec 2014.

[9] O. Kotaba, Nowotsh, Paulitsh J., Stefan M., Theiling M.P., and H. Multicore in real-time systems – temporal isolation challenges due to shared resources. *WICERT 2013*, 2013.

[10] Jure Leskovec and et. al. SNAP Datasets: Stanford large network dataset collection, 2014.

[11] H. Omar, Q. Shi, M. Ahmad, H. Dogan, and O. Khan. Declarative resilience: A holistic soft-error resilient multicore architecture that trades off program accuracy for efficiency. *ACM TECS'18*.

[12] Z. C. Papazachos and H. D. Karatza. Gang scheduling in multi-core clusters implementing migrations. *Future Generation Computer Systems*.

[13] V. Roberge, M. Tarbouchi, and G. Labonte. Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning. *IEEE Trans. on Industrial Informatics*, 9(1):132–141, 2013.

[14] Q. Shi, H. Hoffmann, and O. Khan. A cross-layer multicore architecture to tradeoff program accuracy and resilience overheads. *IEEE Computer Architecture Letters*, 14(2):85–89, July 2015.

[15] Q. Shi, H. Omar, and O. Khan. Exploiting the tradeoff between program accuracy and soft-error resiliency overhead for machine learning workloads. *CoRR*, abs/1707.02589, 2017.

[16] J. Son and J. Alves-Foss. Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems. In *2006 IEEE Information Assurance Workshop*, pages 361–368, June 2006.

[17] P. Subramanyan, R. Sinha, I. Lebedev, S. Devadas, and S. A. Seshia. A formal foundation for secure remote execution of enclaves. In *Conf. on Computer and Communications Security*, 2017.

[18] T. Ungerer, F. Cazorla, P. Sainrat, M. Houston, F. Kluge, S. Metzlaff, and J. Mische. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro'10*.

[19] J. Wadden, A. Lyashevsky, S. Gurumurthi, V. Sridharan, and K. Skadron. Real-world design and evaluation of compiler-managed gpu redundant multithreading. In *ISCA*, pages 73–84, June 2014.

[20] M. Wolf and D. Serpanos. Safety and security of cyber-physical and internet of things systems [point of view]. *Proceedings of the IEEE*, 105(6):983–984, June 2017.