TrustGeM: Dynamic Trusted Environment Generation for Chip-Multiprocessors

Luis Angel D. Bathen, Nikil D. Dutt
Center for Embedded Computer Systems
Donald Bren School of Information & Computer Science
University of California, Irvine
{lbathen, dutt}@uci.edu

***Abstract*—Embedded system security challenges have been exacerbated by the complexity inherent in the software stack of next generation handheld devices (internet connectivity, app stores, mobile banking, etc.) and the aggressive push for multicore technology. As applications with different degrees of assurance are deployed on these multiprocessor platforms, new challenges emerge in terms of protection against software based side channel attacks and exploits such as buffer overruns. In this paper, we introduce TrustGeM: a dynamic trusted environment generation engine for chip-multiprocessors. TrustGeM's goal is to dynamically generate trusted execution environments for applications with different assurance requirements. TrustGeM exploits the concepts of application driven policy generation, performance/power-aware on-chip application sandboxing, and reliable, secure, and dynamic memory virtualization. Experimental results on an 8 Core CMP show that TrustGeM is able reduce overall system energy by an average 24% due to its memory utilization efficiency while incurring minimal performance overhead over the ideal case (an average of 5%). TrustGeM is also able to generate policies with much smaller memory requirements allowing the dynamic trusted environment generation to enforce the policies much more efficiently.**

information assurance; security; chip-multiprocessors; policy; embedded raids-on-chip; isolation; scheduling

I. INTRODUCTION

As semiconductor manufacturers continue to push for Chip-Multiprocessor technology (e.g., IBM Cell [1], Intel's Multi-core [2], Teraflops Research [3], Single Cloud Computer [4], and Tilera Tile-Gx [5]) new challenges emerge in terms of guaranteeing secure execution of a trusted application. Since multicore platforms are capable of running a series of applications with different assurance requirements [6], even OS instances on each core [4], guaranteeing data confidentiality becomes a major point of concern. Moreover, as systems become more open [7], with the ability to download and run pre-compiled applications, combined with greater on-chip resources and the ability to share resources opens the door to new threats (e.g., side channel attacks [9]) that were not present in the uniprocessor domain, much less in closed systems. As a result, any one of these vulnerabilities may lead the system to (a) run a malicious application that tries to access sensitive data via software exploits (e.g., buffer overflows [8]), or (b) expose private information via side channel attacks [9, 14, 15].

This research was partially supported by NSF Variability Expeditions Award CCF-1029783

In order to prevent software exploits, research efforts have been made in the compiler domain [10-13], which analyze the application's source code and attempt at finding vulnerabilities (e.g., opportunities for buffer overruns due to lack of boundary checks). Hardware monitors [22, 23] have also been proposed to detect execution invariance due to code injection/buffer overruns. There has been effort as well in developing full platform support for secure application implementations [8, 19]. Isolation has also been shown to be effective in providing a secure means to execute trusted applications [20, 21], however, they all require the programmer to fully map the application onto the given platform (e.g., design the application with isolation in mind).

In this paper, we propose **TrustGeM**, a dynamic trusted environment generation engine for chip-multiprocessors. The goal of TrustGeM is to build a trusted environment for the execution of trusted applications concurrently with untrusted applications. TrustGeM exploits the ideas of application driven policy generation, on-chip application sandboxing (isolation), and Secure, Reliable, and dynamic memory Virtualization (**SeReVral**) technology.

The main contributions or this paper are:

- SeReVral: Secure, reliable, and dynamic memory virtualization support for CMPs
- SeReVral-aware policy generation
- SeReVral-aware real-time on-chip application sandboxing.

II. TRUSTGEM

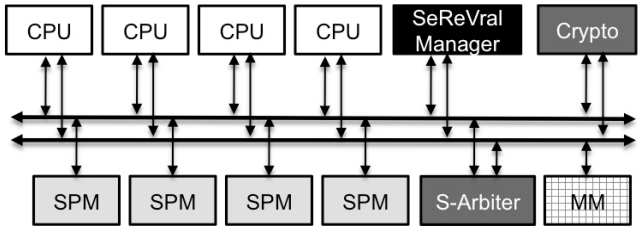

Figure 1. Secure CMP Platform Support

A. Chip-Multiprocessor Platform

Figure 1 shows a high-level block diagram of our proposed platform. It consists of a series of simple RISC cores, a set of distributed on-chip scratchpad memories, a secure arbiter, a crypto engine, and the SeReVral manager module. It consists of an on-chip shared bus (e.g., AMBA AHB bus protocol)

used for internal transactions. All masters in the system can talk to the off-chip main memory (MM) through the off-chip bus. The main difference from this platform to that presented in [6] is the on-chip SeReVral manager. The Crypto co-processor is responsible for providing support for data encryption/decryption. On-chip data is stored in clear text, and any sensitive data coming in/going out of the chip is encrypted/decrypted. This model is standard as sensitive data (e.g., DRM keys) are stored in main memory encrypted, and decrypted (kept in the clear) while being processed by the CPUs. Our SeReVral module is responsible for providing secure and reliable virtual memory support utilizing physical on-chip resources (SPMs) for the platform. All on-chip data is protected via access control list (ACL) enforcement, as encrypting/decrypting the data in real-time whenever it is needed is not energy/performance efficient. ACLs are tied to the hardware IDs associated with each master in the system in order to prevent spoofing of ACLs.

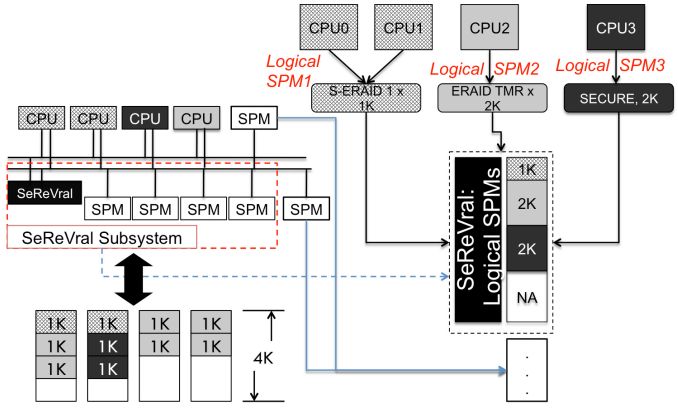

Figure 2. SeReVral Logical SPM Mapping

III. SeReVral: Secure, Reliable, and Dynamic Memory Virtualization

The concept of Embedded RAIDs-on-Chip (E-RoC) was first introduced in [17] with the goal of providing highly reliable memory management for CMP platforms with distributed SPMs. The goal of a traditional RAID system in storage systems is to guarantee the uptime of the system. In case a disk goes bad, the remaining disks are used to 1) serve data requests despite the failed disk, and 2) on disk replacement, rebuild the RAID system. Unlike traditional RAID systems, where disks are replaced following a disk failure, the goal of an Embedded RAID (E-RAID) is to guarantee the validity of the data stored in the E-RAID. Thus custom E-RAID levels were designed for the use in embedded SoCs. Since RAID system parity can be computed by simple XORs, E-RAIDs incur much less performance overheads compared with any of the ECC/hybrid schemes previously proposed [17]. Moreover, E-RAIDs exploit aggressive voltage scaling of memories in order to significantly reduce the power consumption at the cost of exponentially increasing the error rate in memory cells (these errors are automatically handled by the E-RAID levels).

One of the key features in E-RoC is the ability to generate on demand Logical SPMs (LSPMs). Much like the notion of Logical Volumes, Logical SPMs are exposed to the host as regular scratchpad memories, and hence, their management is no different from managing regular scratchpad memories. As shown in Figure 2, the SeReVral module is responsible for creating a virtualized memory space that is viewed by the outside world as regular memory mapped SPMs, however, internally, data is mapped to the distributed physical memories in various forms. Each LSPM is capable of supporting different degrees of reliability and security, from simple mirroring to highly reliable TMR, as well as secure memory space. Note that it is possible to create secure and reliable LSPMs as any E-RAID level can be configured to enforce ACLs. Figure 2 shows the SeReVral memory subsystem with three Logical SPMs (Logical SPM 1-3), where each LSPM guarantees a different degree of reliability and trust. LSPM 1 configured to provide trustworthy memory space for CPUs 0 and 1 through ACL enforcement and running an E-RAID 1 level of 1 KB (mirroring). Logical SPM 2 was configured to provide highly reliable memory through E-RAID TRM for CPU 2, and finally, Logical SPM 3, which provides secure and dedicated memory space for CPU 3. In case of E-RAID creation with ACL enforcement enabled, every memory transaction is validated against the ACLs, and only the E-RAID creator can delete it or update the ACL. Our software API for E-RAID manipulation provides designers with a very small TCB, which removes all E-RAID packet generation from the user's hands, and exposes a very abstract C-like interface (e.g., malloc()). Data within the physical memories is zeroed after E-RAID deletion. Due to space limitation we give a very high level overview of E-RoC, for more information please refer to [17].

```
cpu_0 accesses:      720  162  0
cpu_0 tasks:    3    time:9539
SCHED:  0    1643t2
      MAP:    t2    buf_5      mem_2
            ACL: ( t2:rw t3:r )
      MAP:    t2    buf_8      mem_1
            ACL: ( t2:r t3:rw t4:r )
      MAP:    t2    buf_9      mem_1
            ACL: ( t2:rw )
```

Figure 3. Security Policy Example

IV. Trusted Environment Generation

A. SeReVral-Aware Policy Generation

Custom policy generation presented in [6] assumes that the underlying hardware can dedicate enough resources for the application to execute; thus, each policy contains both task and data mapping information. As we can see from Figure 3, the policy assumes that the underlying hardware can dedicate two memories for the execution of the application. This approach works well when the system is able to support said policies, however, dedicating full memories to support the execution of a trusted application may lead to underutilization of the memory resources, and as a result, poor overall system performance. TrustGeM solves this limitation by rearranging the address mapping of buffers and grouping them into continuous trusted/untrusted memory space; This allows TrustGeM to estimate the maximum required memory needed to support the buffers, thereby providing the system with much more accurate memory requirements information. During the policy generation (scheduling and placement), we replace memories of fixed size (as was previously done) with logical LSPMs of varying sizes leading to better memory utilization.

TABLE I. POLICY SELECTION

	High Load	Low Load
On Battery	Policy 1	Policy 1/2
On Power Cord	Policy 2	Policy 3

B. *Selective Policy Enforcement*

Each application will be bundled with a set of policies; each will have a different execution profile consisting of energy consumption, performance (latency), and resource requirements. Table I shows an example of the selection among three different policies: Policy 1 is a low power, high latency policy. Policy 2 is a mid power/mid latency policy. Policy 3 is high performance, high power consumption policy). As we can see, given the system's load (and status) we can then choose the right policy. Current implementation limits up to three policies per application as we are yet to fully investigate the impact of the policies' footprint on the total size of the (downloadable) application.

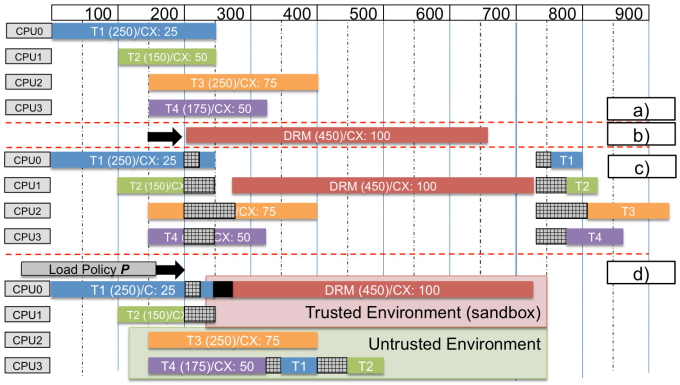

Figure 4. On-Chip Sandboxing Comparison

C. *On-Chip Dynamic Sandboxing*

TABLE II. POLICY REQUIREMENTS

	# Processors Needed	Memories (Sizes)
Application 1	2	3 (2KB, 3KB, 3KB)
Application 2	1	2 (4KB, 2KB)
Application 3	1	1 (2KB)

Figure 4 shows the process of on-chip sandboxing. At first, the system (a 4CPUx4KB SPMs CMP) is running an application (Figure 4 (a)), at time 200; a DRM application wishes to launch (Figure 4 (b)). The traditional halt approach (Figure 4 (c)) will first stop execution of all tasks, context-switch them, start execution of the DRM application, and resume execution of all other tasks after DRM completes its execution. This approach is very secure as it grants full access to the hardware to the DRM system, but it suffers from great performance degradation (on average, a 460 ms delay in execution). Figure 4 (d) shows TrustGeM's sandboxing approach, which first selects and loads a policy for the given system load, it then proceeds to sort all tasks based on their context-switch overhead, and context-switches the ones with the lowest overhead. Next, a call for SeReVral LSPM generation is made (dark box), and execution of DRM is initiated on the given sandbox, while the remaining tasks continue their normal execution on the available CPUs. This approach incurs an average of 90 ms delay (5x much more efficient).

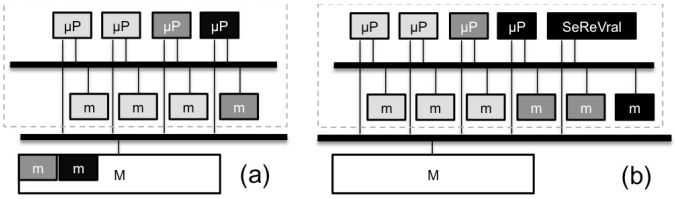

Figure 5. Improved Sandboxing

Figure 5 shows a comparison between sandboxing support for multiple trusted applications (defined in Table II) given PoliMakE (Figure 5 (a)) and TrustGeM (Figure 5 (b)). As we can see, PoliMakE is forced to map data from Applications 1 and 2 (2 x 2KB) onto off-chip memory space, leading to extra off-chip access penalties, affecting both energy and performance of the system. TrustGeM, however, is able to generate enough LSPMs (a total of 6 LSPMs with varying sizes and assurance guarantees) for all applications, therefore improving on-chip memory utilization.

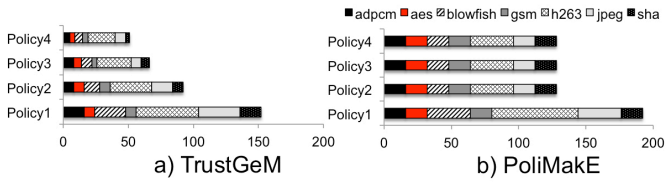

Figure 6. Data utilization for various policies

V. EXPERIMENTAL RESULTS

We analyzed CHStone and Mediabench II benchmarks and generated four different policies with different memory/ performance requirements. We integrated TrustGeM into our simulation environment [6], and added the SeReVral support. We then tested the environment generation on an 8CPU x 8x16KB SPMs @ 65 nm (CMP_L), and a smaller CMP with 8 Cores and 8 KB SPMs (CMP_S). We then evaluated three trusted environment generation mechanisms (Halt [24], PoliMakE [6], and TrustGeM). TrustGeM (like PoliMakE) will outperform the Halt approach as it halts execution of all tasks in order to provide the trusted application with full access to the underlying hardware. Both PoliMakE and TrustGeM generate dynamic trusted environments, however, TrustGeM overcomes PoliMakE's limitations by virtualizing the on-chip memories.

A. *TrustGeM Memory Utilization Efficiency*

The main contribution of this paper is the ability to generate policies with reduced memory footprint and dynamically generate trusted execution environments with efficient memory virtualization (exploiting SeReVral). Figure 6 shows four policies generated with SeReVral (Figure 6 (a)) support and PoliMakE support (Figure 6 (b)). From this figure we can observe how policies generated for TrustGeM have a smaller footprint than PoliMakE policies (up to 81% higher efficiency). TrustGeM is able to accommodate all 7 applications isolated

utilizing only 40% of the available resources. Finally, if all applications are to run concurrently with the policy with highest requirements (Policy 1), we can see that TrustGeM is able to execute them successfully, whereas PoliMakE is forced to map data to off-chip memory.

TABLE III. PERFORMANCE AND ENERGY COMPARISON

Apps	Expected	Halt	PoliMakE	TrustGeM	Energy Savings
adpcm	1	102.25	2.08	1.02	73.86%
aes	1	142.93	1	0.97	19.00%
gsm	1	231.63	1	1.07	3.79%
h.263	1	1	1.09	1.04	14.24%
jpeg	1	6.10	0.98	1.062	19.03%
sha	1	19.46	1	1.13	14.85%
avg	1	83.89	1.19	1.05	24.13%

B. *Policy Enforcement Comparison Under Constraints*

Table III shows the performance comparison among various trusted environment generation schemes (Halt [24], PoliMakE [6], and TrustGeM) and the Expected (ideal) execution of each application assuming no delays due to context switching. For this experiment, we ran the applications in a resource-constrained environment (CMP_S). We started h.263 at t=0, adpcm at t=100, jpeg and gsm at t=300, and sha/aes at t=500 (all K cycles). In the PoliMakE scenario, the first two applications utilized the entire system resources and loaded Policy 1 each. The remaining applications loaded and enforced Policy 2. Each time a new application starts, PoliMakE generates a trusted environment (sandbox) for its execution. PoliMakE suffers from higher data eviction rate as it reserves entire memories for execution of tasks, whereas TrustGeM is able to dynamically generate LSPMs (of various sizes), and thus utilizes the on-chip resources much more efficiently (leading to better energy efficiency and system performance). In general, TrustGeM is able to outperform PoliMakE and remain within the Expected execution (minimal delays). Note that in [6], PoliMakE was shown to reduce latency over the halt approach by 61% and reduce power consumption by 99%. As expected, the halt approach is the worst for all except *h.263*, which is the very first executed application. For this experiment, no application was given higher priority than the rest since we wanted to test how well each scheme was able to accommodate concurrent (sandboxed) execution of all applications as they enter the system.

C. *Energy Savings over PoliMakE*

SeReVral allows TrustGeM to generate policies with lower memory footprints than PoliMakE; combined with SeReVral's aggressive voltage scaling of the memories, TrustGeM achieves higher energy efficiency than PoliMakE. As shown in Table III, we observe up to 73% energy savings for *adpcm* alone, and a system wide energy savings of 24%.

VI. CONCLUSIONS

In this paper, we presented **TrustGeM**, a dynamic trusted environment generation methodology for chip-multiprocessors capable of building a trusted environment for the execution of trusted applications concurrently with untrusted applications. TrustGeM exploits the ideas of application driven policy generation, on-chip application sandboxing (isolation), and Secure, Reliable, and dynamic memory Virtualization (**SeReVral**) technology. Experimental results on an 8 Core CMP show that TrustGeM is able reduce overall system energy by an average 24% due to its memory utilization efficiency while incurring minimal performance overhead over the ideal case (an average of 5%).

REFERENCES

[1] The Cell project at IBM Research, http://www.research.ibm.com/cell/

[2] Intel Multi-Core Technology, http://www.intel.com/multi-core/index.htm?iid=tech_as_lhn+multi

[3] Intel's Teraflops Research Chip, http://techresearch.intel.com/articles/Tera-Scale/1449.htm

[4] Tilera's Tile Gx Family, http://www.tilera.com/products/processors/TILE-Gx_Family

[5] Intel's Single Cloud Computer, http://techresearch.intel.com/ProjectDetails.aspx?Id=1

[6] L. Bathen et al., "PoliMakE: a policy making engine for secure embedded software execution on chip-multiprocessors," *WESS '10*

[7] Android OS, http://www.android.com/

[8] J. Coburn et al., "SECA: security-enhanced communication architecture." In *Proc. of CASES '05.*

[9] Z. Wan et al., "New cache designs for thwarting software cache-based side channel attacks." In *Proceedings of ISCA '07*

[10] G. Necula et al., "CCured: type-safe retrofitting of legacy software." *ACM Trans. Program. Lang. Syst.* 27, 3 (May. 2005), 477-526.

[11] H. Ozdoganoglu et al., "SmashGuard: A Hardware Solution to Prevent Security Attacks on the Function Return Address." *IEEE Trans. Comput.* 55, 2006

[12] C. Cowan, et al., "PointguardTM: protecting pointers from buffer overflow vulnerabilities." In *Proc. of USENIX Security '03*

[13] C. Cowan et al., "StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks." In *Proc. USENIX Security '08*

[14] P. C. Kocher et al., "Differential Power Analysis." In *Proc. of the 19th Annual Int. Cryptology Conference on Advances in Cryptology*, 1999

[15] D. J. Bernstein. Cache-timing attacks on AES. Technical report, 2005. URL: http://cr.yp.to/antiforgery/cachetiming- 20050414.pdf

[16] E. Biham et al., "Differential Fault Analysis of Secret Key Cryptosystems." *In Proc. of CRYPTO '97*

[17] L. Bathen et al., "E-RoC: Embedded RAIDs-on-Chip for Low Power Distributed Dynamically Managed Reliable Memories," DATE '11

[18] Open Mobile Alliance, "*DRM Specification V2.0*", Open Mobile Alliance Ltd, 2004, La Jolla (CA), USA

[19] R. York, A New Foundation for CPU Systems Security. ARM Limited (http://www.arm.com/armtech/TrustZone?OpenDocument), 2003

[20] Shimizu, K., Hofstee, H. P., and Liberty, J. S. 2007. Cell broadband engine processor vault security architecture. *IBM J. Res. Dev.* 51, 5

[21] G. E. Suh et al, "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing," in Proc. Intl Conf. Supercomputing (ICS '03)

[22] K. Patel et al., "SHIELD: a software hardware design methodology for security and reliability of MPSoCs." In *Proceedings of DAC '8*

[23] Z. Shao et al., "Security Protection and Checking for Embedded System Integration against Buffer Overflow Attacks via Hardware/Software." *IEEE Trans. Comput.* 55, 4 (Apr. 2006), 443-453

[24] J. McCune et al., "Flicker: an execution infrastructure for tcb minimization." In *Proc. of EuroSys '08*