Dynamic Security Domain Scaling on Symmetric Multiprocessors for Future High-End Embedded Systems

INOUE, Hiroaki[†], Akihisa Ikeno[‡], Tsuyoshi Abe[†], Junji Sakai[†], and Masato Edahiro[†]

[†] System IP Core Research Laboratories, NEC Corporation
1120, Shimokuzawa, Sagamihara,
Kanagawa, 229-1198 Japan

[‡] NEC Informatec Systems, Ltd.
3-2-1, Sakado, Takatsu-ku, Kawasaki,
Kanagawa, 213-0012 Japan

{h-inoue@ce, a-ikeno@pb, t-abe@dr, jsakai@bc, eda@bp}.jp.nec.com

ABSTRACT
We propose a method for dynamic security domain scaling on SMPs that offers both highly scalable performance and high security for future high-end embedded systems. Its most important feature is its highly efficient use of processor resources, accomplished by dynamically changing the number of processors within a security domain in response to application load requirements. Two new technologies make this scaling possible without any virtualization software: 1) self-transition management and 2) unified virtual address mapping. Evaluations show that this domain control provides highly scalable performance and incurs almost no performance overhead in security domains. The increase in binary code size is less than 40KB, and the time required for individual state transitions is of a single-millisecond order. This scaling is the first in the world to make possible dynamic changing of the number of processors within a security domain on an ARM SMP.

Categories and Subject Descriptors
C.1.4 [**Processor Architectures**]: Parallel Architectures – *mobile processors*; D.4.7 [**Operating System**]: Organization and Design – *real-time systems and embedded systems*.

General Terms
Design, Security.

Keywords
SMP, AMP, Dynamic security domain scaling

1. INTRODUCTION
Future high-end embedded systems, such as mobile phones, digital home appliances and car infotainment systems, will require heavy CPU-centric applications that employ high-load functions (*e.g.*, XML processing, navigation, speech search and speech translation) in order to provide more user-friendly services. This means that embedded processors will have to offer highly scalable performance in response to application loads. One promising approach would seem to be the use of Symmetric Multi-Processors (SMPs), such as ARM MPCores [1]. The use of SMPs would help exploit both the thread-level and process-level parallelism of applications and would achieve flexible dynamic load distribution over the processors as a whole.

Future high-end embedded systems will also require a mechanism to execute native applications downloaded from open networks in order to provide high flexibility for users. This means that the security needed to protect pre-installed (*i.e.*, basic-function) applications will become an increasingly important issue since such downloaded native applications might include viruses. An especially important security technique is the creation of OS instances, called security domains. A security domain is specifically defined as an isolated execution environment prepared for a group of applications. Security domain isolation makes it possible to prevent illegal access to the address spaces of other security domains and to limit the maximum amount of resources that applications on the security domain may use. Intel and NTT DoCoMo have, in fact, jointly announced new mobile phone specifications [8], referred to as the Open and Secure Terminal Initiative (OSTI), that are designed to make possible the installation of a wide variety of OSs and applications with the use of security domains. One particularly promising approach for supporting security domains is the use of Asymmetric Multi-Processors (AMPs) [6] [7], which would help enhance system security since this approach enables individual security domains (OSs) used for pre-installed or downloaded applications to be independently executed on each processor of the AMP with support of hardware designed for processor-level separation. This means that it will be highly secure to be able to protect pre-installed applications on processors from interference from applications downloaded to other processors by means of processor-level separation. This approach would also enhance application performance because no virtualization software is required [2] [3] [4] [13].

Neither the SMP nor the AMP approach is, however, in itself satisfactory. While the SMP approach provides highly scalable performance for heavy pre-installed applications by means of dynamic load distribution over the processors as a whole, it fails to support the multiple processor-level security domains required to enhance system security. This is because the SMP OS manages all processors contained in an SMP in order to provide high scalability. By way of contrast, the AMP approach supports the multiple processor-level security domains required to enhance system security by means of processor-level separation. Unfortunately, however, this separation fixes the number of processors within an OS. Thus, this AMP approach results in providing heavy pre-installed applications with only low performance scalability.

This paper reports our design and implementation of dynamic security domain scaling on SMPs in order to provide both highly scalable performance and high security in future high-end embedded systems. Our design is a hybrid of the SMP and AMP approaches. Where the execution of heavy pre-installed applications is required, our dynamic security domain scaling enables all processors contained in an SMP to be allocated for pre-installed applications by means of the SMP approach. Where coordination between pre-installed and downloaded applications is required in order to provide system flexibility, our dynamic domain security scaling reduces the number of processors allocated for pre-installed applications and allows deallocated processors to be dynamically allocated for the execution of downloaded applications. This achieves processor-level separation of security domains by means of the AMP approach.

The major contributions of this work include achievement of the following design objectives:

- **Flexible Security Domain Scaling:** The number of processors within a security domain must be flexibly changeable in order to provide scalable performance for pre-installed or downloaded applications. By integrating a conventional CPU Hotplug technology [12] with our innovative context handling technology, our dynamic security domain scaling satisfies this requirement without any virtualization software.
- **Highly Scalable Performance:** Future high-end embedded systems will be required to execute heavy pre-installed applications in order to achieve user-friendly services. The SMP element of our approach provides highly scalable performance in pre-installed applications since the use of a SMP achieves flexible dynamic load distribution over the processors as a whole.
- **Hardened Security:** Pre-installed applications executed in high-end embedded systems must be protected from downloaded native applications. With the AMP element of our approach, the creation of a separate, processor-level domain for pre-installed applications helps ensure their greater protection.
- **High Performance in Security Domains:** Since our non-virtualization approach provides physical processors for pre-installed and downloaded applications, it enhances application performance.
- **Small Binary Code Size:** Future embedded systems will need to be able to operate with limited resources, and the integrated design of our dynamic security domain scaling is designed to help them operate with a smaller binary code size.

Our dynamic security domain scaling features two new technologies: 1) self-transition management, by which processors for pre-installed applications are dynamically allocated for the execution of downloaded applications; and 2) unified virtual address mapping, by which there are seamless transitions back and forth between the executions in separate security domains, those for downloaded applications and those for pre-installed applications. These technologies help make SMPs with our dynamic security domain scaling ideally suited to high-end embedded systems.

The remainder of this paper is structured as follows: Section 2 describes related work, Section 3 explains our dynamic security domain scaling, Section 4 shows the results of our evaluation of it, and Section 5 summarizes our work.

2. RELATED WORK

Our research differs in a number of respects from the current body of research on domains. Our dynamic security domain scaling is designed to exploit the benefits of both AMPs and SMPs.

Major user-level domain approaches include eSOL eT-Kernel [5] and QNX BMP [11]. Since these approaches use processor affinity settings to allow applications executed on an SMP OS to be run only on specified processors, they make it possible to change the number of processors assigned to an application. The execution of pre-installed and downloaded applications on the same SMP OS, however, will result in critical security vulnerability.

Major kernel-level domain approaches, such as SELinux [9], allow both pre-installed and downloaded applications to be run on the same SMP OS since a security module within the SMP OS monitors system calls issued from all applications and imposes mandatory access control on all applications. Such monitoring results in severe performance degradation, however, even in pre-installed applications. Further, with such approaches, it is difficult to avoid security vulnerability in OSs and security modules.

Major virtualized domain approaches include type-I VMMs, such as LPAR [2] and Xen [3], and type-II VMMs, such as UML [4] and VMware [13]. VMMs enhance system security since they allow pre-installed applications to be separated from downloaded applications at the OS level. In addition, VMMs make it possible to provide any number of processors to pre-installed applications through processor virtualization features. Virtualization, however, results in unignorable performance degradation, and there is a degree of security vulnerability in complex virtualization software.

By way of contrast, SMP platforms with our dynamic security domain scaling not only allow the number of processors within a security domain to be dynamically changed, they also make it possible to support high security by means of processor-level separation, and application performance in security domains is high because no virtualization software is required.

3. SMP PLATFORM WITH DYNAMIC SECURITY DOMAIN SCALING

3.1 Overview and Principles

Figure 1 shows the application of our dynamic security domain scaling to an SMP platform containing four processors. This platform has three security domains, a base domain for the execution of pre-installed applications and two security domains (A/B) for the execution of downloaded applications. It contains a function for changing the number of processors assigned to individual security domains. The base domain maintains at least one processor for its executions. Note that our scaling is not limited to use with the structure shown in this figure; any number of processors can be dynamically allocated to any number of security domains with any kind of OSs.

For heavy pre-installed applications, all four processors are allocated to the base domain on the SMP OS. Further, where coordination is required between pre-installed and downloaded applications, a processor allocated to the base domain (*e.g.*, CPU#3) will be yielded to a security domain used for downloaded applications. As shown on the right-hand side of the figure, pre-installed applications can, for example, be executed separately on an SMP OS with three processors, while downloaded applications are executed with the remaining processor. Note that an domain manager application, which supports the similar functions of an Java application manager, is executed on the base domain in order to control the timing of domain switching and the allocation of processors to individual domains.

The platform contains both a software component, which includes a context manager and context handlers, and a hardware component, which includes bus filter logic. Here, context refers to the register values required to restore a processor state (*e.g.*, mode registers and CP15 control registers of an ARM processor).

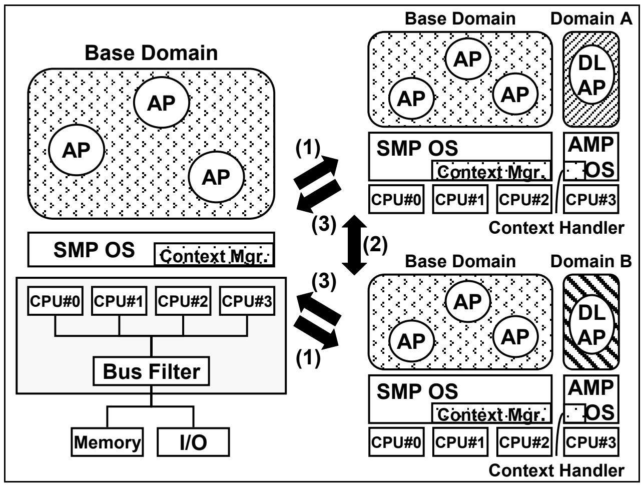

Figure 1: SMP Platform Applied to Dynamic Security Domain Scaling

The context manager is run only on the base domain, and it manages base domain contexts, which are required to restore to the base domain any processors previously allocated to security domains. It also manages all security domain contexts for downloaded applications. Further, it controls dynamic security domain scaling (see Section 3.2.1). It also sends to a context handler the context of a security domain in which an execution is to be performed and orders that the execution be made.

A context handler [7] is run on each security domain for downloaded applications, and it conducts domain switching, from a current security domain to a security domain specified by the context manager. To do this, it functions as an interrupt handler.

In transitions from, for example, a state with only a base domain to one with both a base domain and a security domain (*e.g.*, domain (A) in state transition (1) in Figure 1), the context manager saves the context of the processor (here, CPU#3) allocated for the execution of the security domain (A), and restores the context of that security domain to the processor. As a result, the number of processors allocated to the base domain is dynamically reduced from four to three.

In state transition (2) in Figure 1, *e.g.*, in switching from security domain A to security domain B, the context manager sends the context of security domain B to the context handler of security domain A, using shared memory and inter-processor interrupts in inter-processor communication, and it receives from the context handler the context of security domain A, which had previously been saved. As a result, switching from security domain A to security domain B is seamless.

Finally, in state transition (3) in Figure 1, *e.g.*, in switching from a state with both a base domain and a security domain to one with only a base domain, the context manager sends the base domain context of the processor performing executions in security domain B (*i.e.*, CPU#3) to the context handler of security domain B, and it receives from the context handler the context of security domain B, which had previously been saved. As a result, the processor allocated for executions in a downloaded domain (*i.e.*, CPU#3) is restored to the base domain, and the number of processors allocated to the base domain is dynamically increased from three to four.

Bus filter logic [6] determines whether an access from a processor to a bus slave should be granted. This decision is made on the basis of an access matrix. While the processors executing in the base domain, for example, are allowed access to any resources, such as I/Os or memories, processors executing in security domains for downloaded applications are allowed access only to resources allocated to a specific security domain, and are prohibited to access to resources used in an other domain. In this way, this platform achieves hardened protection among security domains at the processor level. In addition, in contrast to the slow access checking offered by such virtualization software as type-I or type-II VMMs, this logic offers the fast access checking required to maintain separate security domains.

3.2 Two New Technologies

In order to apply our dynamic security domain scaling to an SMP platform, we had to address two important technical issues, particularly with respect to the important state transitions of (1) and (3) in Figure 1: 1) how processors to be used for downloaded applications are to be dynamically separated from the base domain and then merged again with it; and 2) how the execution of a software component to be shared among domains, including the base domain, is to be stabilized during state transitions between different OSs. In the following sections, we assume that an ARM MPCore [1], which provides both AMP and SMP modes to each processor, is used as an SMP, and that Linux is used for the OSs of security domains.

3.2.1 Self-Transition Management

For separating a processor from a base domain (state transition (1) in Figure 1) and merging a processor back to the base domain (state transition (3) in Figure 1), simple operational control of processors, such that including only *suspend* and *resume*, would be insufficient. In addition to operational control, the base domain is also required to support context handling for the processors.

Self-transition management achieves such context handling for processors by means of integrating CPU Hotplug technology [12] with it. CPU Hotplug technology, originally developed by Russell et al., is used to remove faulty processors from a system and add new processor substitutes to that system without stopping on-line operations. In ARM MPCore Linux, this technology allows unused processors to be put into a low power mode in order to reduce power consumption. In other words, it simply suspends use of the processors.

Figure 2 shows the relationship between our self-transition management and CPU Hotplug technology. In the figure, gray boxes indicate newly-added operations for the self-transition management. With respect to CPU Hotplug technology (*i.e.*, the white boxes) implemented on ARM MPCore Linux, when a processor in an idle state is put into a low power mode, the CPU Hotplug technology requests the execution of a "CPU Hot Remove," which might involve, for example, 1) the migration of processes previously executed on that processor to other live processors, 2) a change in interrupt distribution to the processor, 3) deactivation of cache coherence**,** or 4) a processor's waiting for an interrupt while clock gating is being conducted. After the processor receives a wake-up interrupt, the technology requests the execution of a "CPU Hot Add," such as the re-activation of cache coherence or the return of a processor to an idle state.

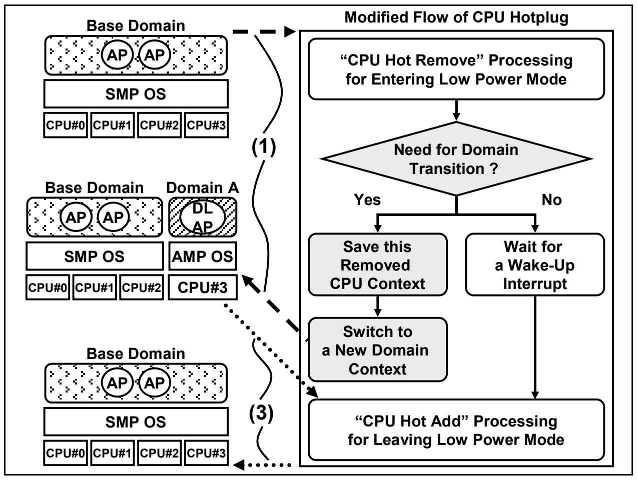
Base Domain
AP
AP
SMP OS
CPU#0
CPU#1
CPU#2
CPU#3
(1)
Base Domain
Domain A
DL AP
SMP OS
AMP OS
CPU#3
Base Domain
SMP OS
(3)
Modified Flow of CPU Hotplug
"CPU Hot Remove" Processing for Entering Low Power Mode
Need for Domain Transition ?
Yes
No
Save this Removed CPU Context
Wait for a Wake-Up Interrupt
Switch to a New Domain Context
"CPU Hot Add" Processing for Leaving Low Power Mode

Figure 2: Self-Transition Management – Integration between CPU Hotplug and Fast Context Handling

Our self-transition management modifies the operational flow of CPU Hotplug technology. In the case of separating a processor from the base domain and allocating it to a security domain for downloaded applications (*i.e.*, state transition (1) in Figure 1), it requests the processor to execute a "CPU Hotplug Remove." After that, rather than make the processor wait for an interrupt, it saves the base domain context required to restore the processor and restores to the processor the context of the security domain for downloaded applications. In this way, our self-transition management enables processors which previously executed functions in the base domain to start to execute in security domains. *The key feature in our self-transition management is changing the value of the program counter saved in a base domain context to the program address which corresponds to the point at which waiting for an interrupt has been completed, i.e., the address that corresponds to the point just before "CPU Hot Add" processing commences.*

In the case of merging a processor from a security domain back into the base domain (*i.e.*, state transition (3) in Figure 1), the self-transition management requests the context manager to perform a domain context switch. Using the base domain context required to restore the processor to the base domain, the context manager orders the processor's context handler to perform a domain context switch. The context handler then conducts a domain switch from the current security domain to the base domain. Here, as mentioned earlier, since the value of the program counter is changed to the address directly preceding "CPU Hot Add," the processor executes "CPU Hot Add" and returns to an idle state in the base domain *as if it had received a wake-up interrupt*. In this way, the self-transition management enables processors which previously executed functions in security domains to resume making executions in the base domain.

Note that, while our self-transition manager is not based on virtualization software technologies, the security level of domain separation is kept high by means of bus filter logic. Thus, SMP platforms with dynamic security domain scaling are able to provide highly secure, high-performance domains.

3.2.2 Unified Virtual Address Mapping

For a state transition between security domains, all registers in a processor have to be set with the register values of a new domain context. Traditional embedded processors, including ARM MPCores, generally do not allow mode registers or control registers, such as a pointer register for use with a page table, to be simultaneously restored. This restriction would make the register setting code executed for state transitions unstable during the register setting between a pointer register setting and the program counter setting, since the register setting code executed in an OS *before* a state transition would use virtual addresses different from those in an OS *after* the state transition.

To avoid this situation, we employ unified virtual address mapping, a technology for matching virtual addresses in the register setting code shared between an OS used before a state transition and an OS used after that state transition. It is employed to help achieve stable state transitions. Figure 3 shows the mapping between physical addresses and virtual addresses in terms of both the SMP OS of the base domain and AMP OSs of security domains used for downloaded applications (*i.e.*, security domains A and B). Unified virtual address mapping arranges common instructions and data used for the register setting code in an area of the physical memory (*e.g.*, 0x0e001000 for the common instructions in Figure 3 and 0x0f000000 for the common data) that is separate from areas of the physical memory used by the SMP OS and AMP OSs. Further, it assigns the common instructions and data to virtual addresses that are the same in both the SMP OS and the AMP OSs (*e.g.*, 0x0ffb0000 in Figure 3). In this way, unified virtual address mapping achieves stable operations, enabling, for example, a processor executing the register setting code to fetch correct instructions or read correct data even after the setting of a pointer register to a page table, since the instructions and data used for the register setting code are assigned to the same virtual addresses as those in both the OS used before a state transition and the OS used after that state transition.

In addition, unified virtual address mapping is designed to prevent extra virtual addresses from being newly allocated to OSs, since it utilizes unused virtual addresses within the virtual address ranges allocated to I/O devices. This means that no extra virtual addresses are required for mapping our register setting code.

Note that bus filter logic makes unified virtual addresses read-only, since the register setting code only fetches instructions and reads data. This results in protecting the code from modifications that might be caused by viruses, and SMP platforms with our dynamic security domain scaling help maintain system security.

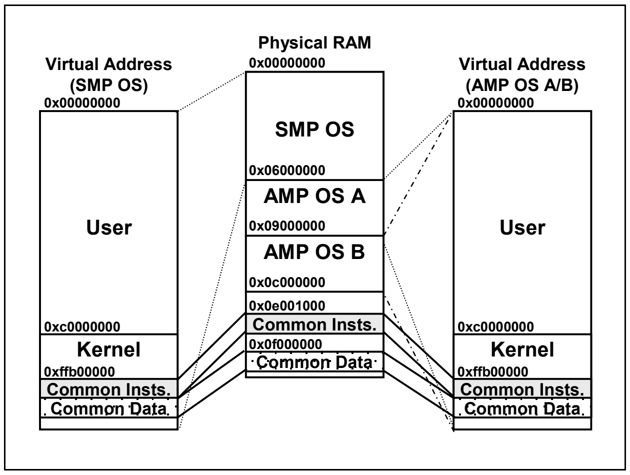

Figure 3: Unified Virtual Address Mapping for Stabilizing Dynamic Security Domain Scaling

4. EVALUATION

Evaluation conditions are summarized in Table 1.

Table 1: Evaluation Conditions

Item	Feature
SoC	MPCore (MP11 CPU x 4) @ 130nm
Cache	I$: 32KB, D$: 32KB per MP11 CPU
Clock Frequency	ARM: 240MHz, Bus: 35MHz
OS	Linux 2.6.7 / SMP OS x 1, AMP OS x 2

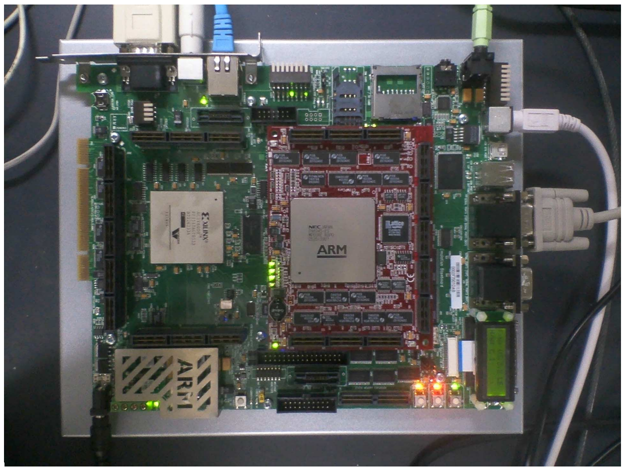

Figure 4: Evaluation Board with an ARM MPCore

4.1 Highly Scalable Performance

Figure 5 shows allocation of MultiProcessor Dhrystone MIPS (MP DMIPS) to the base domain and a security domain in response to dynamic security domain scaling. We have confirmed that state transition (1) in Figure 1 reduces total performance in the base domain by an amount corresponding to that of a single processor, and that the amount of reduced performance is gained in the security domain. Further, with state transition (3) in Figure 1, performance in the base domain is increased back to the previous level.

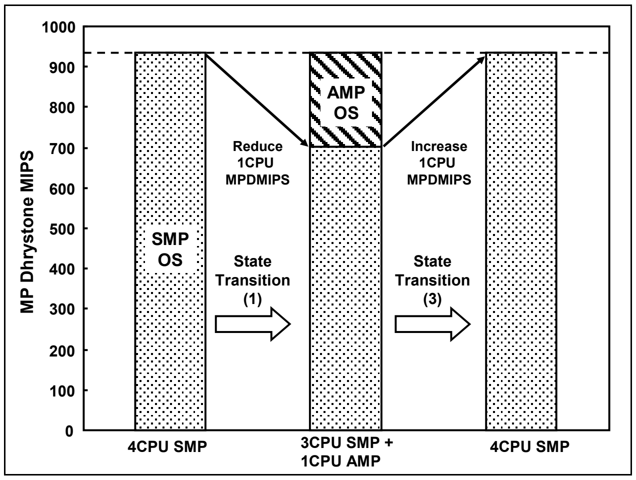

Figure 5: Highly Scalable Performance in the Base Domain

4.2 High Performance in Security Domains

Figure 6 and Figure 7 show the results (*i.e.*, average of 10 measurements) for LMbench [10] processes and context switching micro-benchmarks executed in the base domain. LMbench is a typical OS benchmark. Average results for conventional virtualization software, Xen (a type-I VMM) and UML (a type-II VMM) are also shown in the figure [3]. For the sake of comparison, also shown are performance results normalized to 1 with respect to the reference base of micro-benchmarks executed with unmodified Linux.

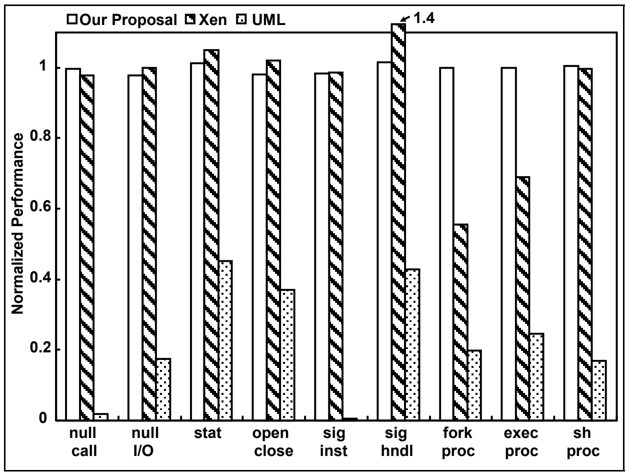

Figure 6: Low Performance Overhead in the Base Domain – Process Micro-Benchmarks -

The base domain with dynamic security domain scaling achieves nearly the same performance as does the base-reference SMP Linux. This cannot be said for conventional virtualization software. Here, our bus filter logic helps provide fast checking of access requests issued from security domains. Two small anomalies seen here, signal handling in Xen and two processes of 16KB array size each with our approach, presumably occurred due to a fortuitous cache alignment (see [3]). Note that we have also confirmed that the performance overhead in security domains for downloaded applications is almost the same as that with unmodified AMP Linux.

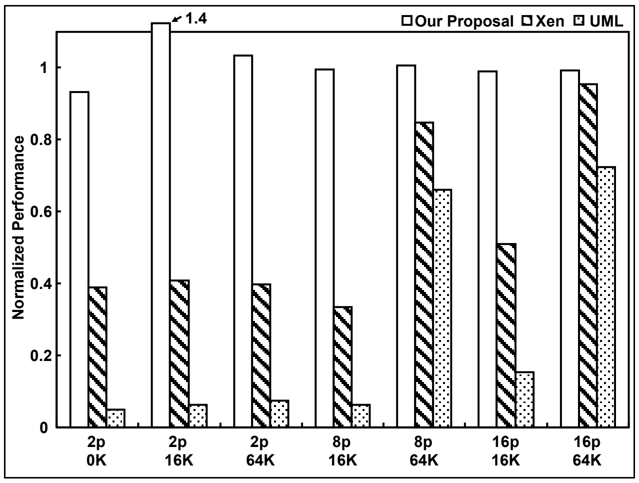

Figure 7: Low Performance Overhead in the Base Domain – Context Switching Micro-Benchmarks -

4.3 Small Binary Code Size

As may be seen in Table 2, dynamic security domain scaling is implemented with a small binary code size (*i.e.*, less than 40KB) by means of the CPU Hotplug integrated design. The binary code size for common text (instructions) and data is also small, being implemented in only 9.2KB. The increases in binary code size of SMP Linux and AMP Linux with our dynamic security domain scaling are only 1.5% and 1.3%, respectively, over that for unmodified OSs. In terms of Lines of Code (LOC), the modified LOC values of SMP Linux and AMP Linux with dynamic security domain scaling are 1549 LOC and 1145 LOC, respectively.

Table 2: Increases in Binary Code Size (KB)

Linux	Text	Data	BSS	Common Text	Common Data	Total
SMP	+11.2	+1.6	+16.2	+0.3	+8.9	+38.2
AMP	+6.6	+1.1	+16.1			+32.9

4.4 Low State Transition Time Overhead

We measured the elapsed time from initiating a state transition request in the base domain to finishing processing the request in an other domain. Table 3 shows times required for the state transitions shown in Figure 1. Values in parenthesis indicate differences in time with respect to corresponding processing using CPU Hotplug technology. The time required for state transitions with dynamic security control is of a single-millisecond order. Further, the greatest time difference with CPU Hotplug technology is only 2.0ms.

Table 3: Low Overhead of State Transition Time

State Transition in Figure 1	Time
(1) Separation from the base domain	2.5ms (+1.5ms)
(2) Switching to a security domain for downloaded applications	0.5ms (---------)
(3) Merge to the base domain	4.5ms (+2.0ms)

5. CONCLUSION

The requirements for more highly scalable performance and higher security in future high-end embedded systems will necessitate the use of SMPs. We have here proposed dynamic security domain scaling on SMPs that allows the number of processors within a security domain to be dynamically changed, and we have applied our approach to ARM MPCores. Key to the success of this dynamic security domain scaling are two new technologies: self-transition management and unified virtual address mapping. We have shown the effectiveness of the scaling in our evaluations with respect to performance scalability, performance overhead, binary code size, and state transition times.

6. REFERENCES

[1] ARM. *ARM11 MPCore Processor Technical Reference Manual*. Revision r1p0, August 2006.

[2] Armstrong, W. J. et al. Advanced Virtualization Capabilities of POWER5 Systems. *IBM Journal of Research and Development*, 49, 4/5(July/September 2005), 523-532.

[3] Barham, P. et al. Xen and the Art of Virtualization. In *Proceeding of the ACM Symposium on Operating Systems Principles,* October 2003, 164-177.

[4] Dike, J. A User-Mode Port of the Linux Kernel. In *Proceedings of the 4th Annual Linux Showcase & Conference,* 2000, 63-72.

[5] eSOL. eT-Kernel Multi-Core Edition. http://www.esol.co.jp/english/embedded/et-kernel_multicore-edition.html.

[6] Inoue, H. et al. FIDES: An Advanced Chip Multiprocessor Platform for Secure Next Generation Mobile Terminals. In *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis,* September 2005, 178-183.

[7] Inoue, H. et al. VIRTUS: A New Processor Virtualization Architecture for Security-Oriented Next-Generation Mobile Terminals. In *Proceedings of the ACM/IEEE Design Automation Conference,* July 2006, 484-489.

[8] Intel, and NTT DoCoMo. *Open and Secure Terminal Initiative (OSTI) Architecture Specification.* Revision 1.00, November 2006. http://www.nttdocomo.co.jp/binary/pdf/corporate/technology/osti/OSTI_Arch_R1_00.pdf.

[9] Loscocco, P. and Smalley, S. Integrating Flexible Support for Security Policies into the Linux Operating System. In *Proceedings of the FREENIX Track of the USENIX Annual Technical Conference,* 2001, 29-42.

[10] McVoy, L. and Staelin, C. lmbench: Portable Tools for Performance Analysis. In *Proceedings of the USENIX Annual Technical Conference,* January 1996, 279-294.

[11] QNX. Multi-Core Technology Development Kit. 2006. http://www.qnx.com/download/download/12449/194.09_Multicore_TDK_p41.pdf.

[12] Russell, R. et al. Linux Kernel Hotplug CPU Support. In *Proceedings of the Linux Symposium,* vol. 2, July 2004, 467-480.

[13] Sugerman, J. et al. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the USENIX Annual Technical Conference*, June 2001, 1-14.