Towards scalable and secure execution platform for embedded systems

Junji Sakai INOUE, Hiroaki Masato Edahiro

System Devices Research Laboratories,
NEC Corporation
1120, Shimokuzawa, Sagamihara, Kanagawa 229-1198, Japan
Tel : +81-42-771-0699
Fax : +81-42-771-0881

jsakai@bc.jp.nec.com h-inoue@ce.jp.nec.com eda@bp.jp.nec.com

Abstract – Reliability of embedded systems can be enhanced by multicore and partitioning approaches. Physical partitioning based on AMP multicore achieves runtime stability of multiple applications in a system and prevents the whole system shutdown as well even when a malicious code creeps in. Combined with logical partitioning by processor virtualization and SMP technologies, the multicore architecture could realize more flexible and more scalable platform for future embedded systems.

I. Introduction

Multicore processors have become popular in enterprise servers and PCs in order to reduce the processors' power consumption, which is now close to maximum limits for the equipment [1]. A shift to multicore is now gradually occurring in embedded systems as well, in which low power with high performance is quite essential. In addition, embedded systems also strongly require operation stability and reliability. Although traditional embedded devices with rather simple functions implemented on realtime OSs have provided these features without difficulty, recent ones, which employ richer OSs such as Windows or Linux to realize more advanced functions, have some challenges in maintaining the features. For example, advanced cell phones are required to maintain their multimedia performance even when several rich applications run simultaneously, and moreover, they should still provide core functions such as telephone calls even when a user innocently downloads and runs an interesting program with a hidden virus. These requirements are unique to embedded devices, and we think that we must establish a special multicore architecture well suited to the characteristics of embedded systems.

In this paper, we introduce physical partitioning to cope with these unique requirements and show two cases where we balance partitioning with communications. Then we present a more flexible partitioning mechanism free from the hardware restriction, and briefly study the coming SMP approach.

II. Reliability and Partitioning

As we mentioned, reliability is considered quite essential in embedded devices. Here, by "reliability" we mean two characteristics.

- Performance Assurance
  Reliability in normal operation. Whatever types of applications are mixed in a single system, executions of all the tasks should be well coordinated so that each application fulfills its performance requirement.
  This challenge arises from the difficulties in application scheduling. It is hard for system designers to control the behavior of many entangled tasks on a complicated OS and layered middleware.

- Robustness
  Reliability in abnormal situations. Safety mechanisms should prevent the whole system from going down, even when some unintentional bugs or some malicious viruses exist in the system. Vulnerable points are unpredictable and likely to be increasing.

There are two approaches to achieve these reliability requirements. These approaches have tradeoffs in performance, certainty, flexibility and cost.

1. Hardware approach:
   Partition the system using a multicore architecture so that interference between the subparts is reduced. The most obvious method is physical partitioning along each processor core boundary.

2. Software approach:
   Reinforce the scheduling mechanisms and enhance the system protection functions. These are typically accomplished mainly by modifications in the OS kernel and its libraries.

Regarding the above reliability issue of embedded systems, we decided to choose the hardware approach for a start, because we considered the certainty of reliability as the primary issue.

III. Partitioning by Physical Cores

A. Multitasking Parallel

Fig. 1 shows our basic structure we have chosen for the hardware approach mentioned above. The point is that we

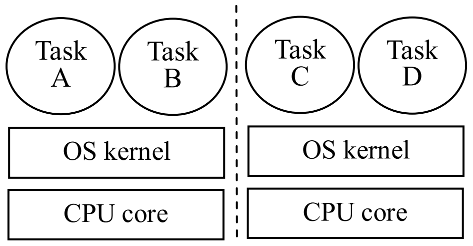


Fig. 1. Multitasking Parallel model.

have adopted both CPU core level- and OS kernel level partitioning. We call the way of the partitioning “Multitasking Parallel.” [2]

In the Multitasking Parallel method, an independent instance of the OS kernel runs on each CPU core in an AMP (Asymmetric Multiprocessing) type multicore LSI. Because task schedulers, who assign CPU resources to the tasks, stay on each OS instance (namely on each physical CPU core) one by one, it is easy to prevent bad influences from one application on a core from affecting another application on another core. For example, an interrupt-driven real-time application would not interfere with a CPU-centric application if they were on different physical cores.

Generally, partitioning and communication are tradeoffs. Communication between applications becomes more difficult if we make the partitioning harder for the sake of good performance assurance. On the other hand, application interference will be present again if we permit unrestricted communications. How to manage the tradeoff between cooperation among cores and potential interference requires further development. Now we present two cases about physical partitioning, and we take this topic in each case.

B. *Performance Assurance by Multicore*

OS wrapper: To evaluate the performance assurance, that is, stability of simultaneous operations of multiple applications, we have to port the applications onto the multitasking parallel environment, where we have to take the inter-task communication into account. Since the OS for each CPU core is based on a conventional OS for a single CPU, the OS supports only local communications within the core. Programmers have to choose the best APIs to use based on which core the peers that will communicate exist on. This is undesirable because the programmers are forced to expand extra effort in order to port their applications to multicore environments.

There are two ways of dealing with the communication issue.

1. Modify the set of communication modules and their functions so that they support inter-core communications as well as intra-core ones.
2. Achieve inter-core communications with compatible APIs by placing a new middleware layer above the OS kernel.

We adopted the latter one since it is more independent from the detailed implementation of the OS kernel. We named the middleware “OS Wrapper” because of its placement to the OS kernel.

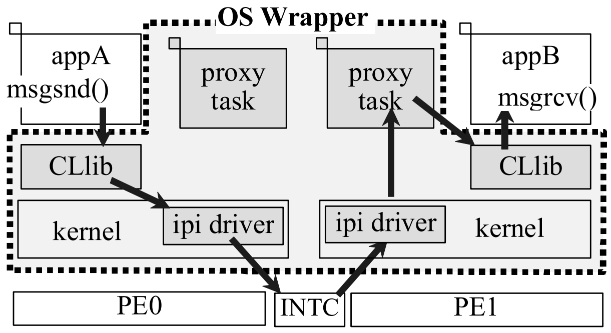


Fig.2 OS wrapper structure

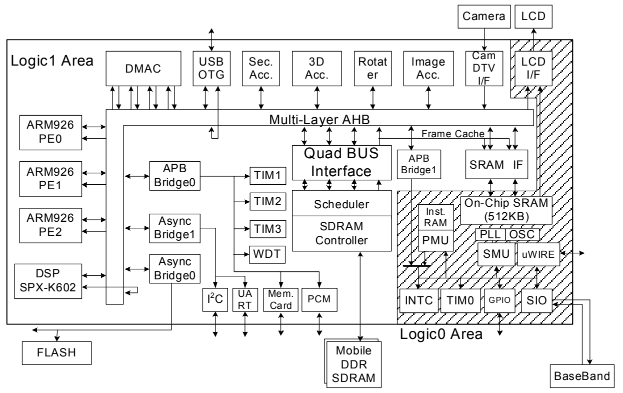


Fig. 3. MP211 chip block diagram.

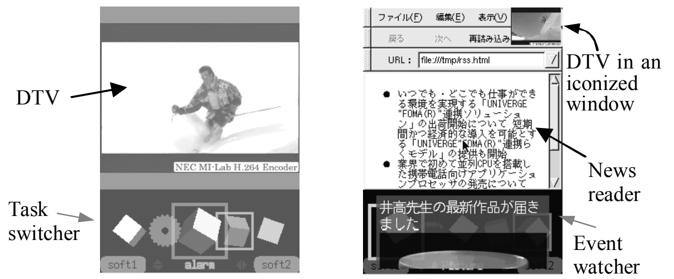
The OS wrapper structure is shown in Fig.2. A client library (CLlib) linked with an application hooks the inter-task communication API calls, and sends messages to the destination through the local interrupt mechanism between cores if the message recipient is on another CPU core. Proxy tasks are placed to wake up the dormant receiver task on behalf of the scheduler in the OS kernel. We designed the API functions of the OS wrapper compatible with those of a single CPU OS, so that applications on any core can communicate with each other using the same API names as those used with traditional single CPU OSs.

Implementation: We implemented the above mechanisms on our multicore SoC MP211 (Fig. 3) [3], which features:

- AMP architecture including 3x ARM9 CPU cores and 1x DSP core
- High performance memory bus architecture to support simultaneous operations of the cores
- Energy saving circuits to reduce operating and sleeping power consumption

Three Linux kernels run on three ARM cores, and the OS wrapper is implemented as a combination of Linux device drivers and user-land libraries linked with applications. Our current implementation supports communication APIs compatible with SystemV IPCs (message queues, semaphores, and shared memory objects) and UNIX domain sockets.

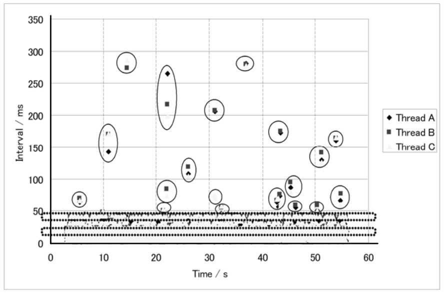
Evaluation: We chose an application set composed of:



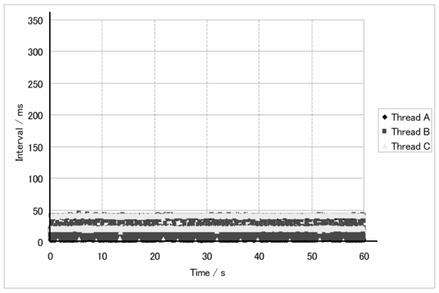
- いつでも・どこでも仕事ができる環境を実現する「UNIVERGE "FOMA(R)"連携ソリューション」の出荷開始について 短期間かつ経済的な導入を可能とする「UNIVERGE"FOMA(R)"連携らくモデル」の提供も開始
- 業界で初めて並列CPUを搭載した携帯電話向けアプリケーションプロセッサの発売について

井高先生の最新作品が届きました

Fig. 4. Screenshots of the evaluation applications



(a) single core execution



(b) multicore execution

Fig. 5. Jitter of task intervals

- a DTV viewer which consists of an H.264 decoder (a DSP task) and its synchronization tasks (CPU tasks)
- a network news reader which consists of an RSS parser and an HTML browser

And we assign the application tasks to the CPU and DSP cores in MP211 as following:

- CPU core0: HTML browser, X server
- CPU core1: RSS parser
- CPU core2: DTV synchronization
- DSP core: H.264 decoder

A screenshot of the application set running on MP211 is shown in Fig. 4. In case we run the application mix on the multitasking parallel environment (i.e. using all three CPU cores), both applications keep running smoothly. By contrast, in case we run them on a single CPU core, we can easily observe short disruptions in the sound playback.

Fig.5 shows jitter of the DTV synchronization task intervals, and we can see obvious improvements in jitter occurrence in the multitasking parallel execution. It shows that stable execution of multiple applications is achieved by partitioning the CPU resources. Note that the above applications are programs for a single CPU environment and we made no modifications on them. This means the OS wrapper achieves a seamless multicore communication environment despite the physically partitioned hardware architecture.

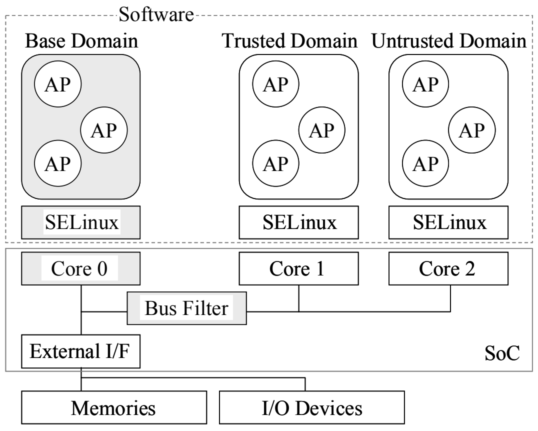


Fig. 6. Domains in FIDES

C. *Robustness by Multicore*

Multicore architecture can be adapted to improve system robustness. Application downloading, which is now quite popular in Java applets for cell phones, is expected to be applied to native binary programs for a wider group of embedded devices in the near future, because native applications have the advantage in run-time efficiency and have the ability to specify detailed actions specialized for the devices. On the other hand, appropriate safeguards are necessary to prevent native applications from accessing unauthorized system functions. Partitioning can be applied to this issue so that the system can maintain proper security.

We developed a multicore security platform called "FIDES" that provides a secure execution environment for downloaded native applications [4]. Its basic concept is to partition the whole system into several domains with different trust levels and to execute each application only in its corresponding domain.

Consider the case of a cell phone. The core functions of a cell phone such as telephone calls, emails and PIM functions (e.g. address books, task lists and personal notes) are executed in the Base domain (Fig. 6). Downloaded applications are assigned to ether Trusted or Untrusted domains, depending on the level of trustworthiness of the applications (e.g. whether it has been validated by the telecommunication carrier or not). This separation prevents unauthorized access to the core functions.

In order to make FIDES more practical, we incorporated a hard-wired filtering logic called Bus Filter with the multitasking parallel architecture. The bus filter mechanism prohibits unauthorized access from less trusted domains to the main memory and/or I/O devices that are shared among the domains. Thus we partitioned the memory and I/O resources as well as CPU resources.

The issue of how to best balance partitioning and communications still exists in this area. There should be communication paths between domains to communicate, for example, the downloaded contents and the DRM (digital rights management) functions. Malicious activities could propagate over the paths into the base domain. We introduced a mechanism called Dynamic Access Control, which restricts the access rights of a task dynamically when it talks with a less trusted task.

We implemented the FIDES platform with MP211 and SELinux (Security-Enhanced Linux), and our evaluation results show that the bus filter logic size is estimated to be small enough in the total die area (<1%), and that performance overhead due to the dynamic access control is only 4%. Thus we can say that partitioning can provide secure environment, with necessary communications granted.

To summarize, we have shown that the physical partitioning with an AMP multicore can achieve the reliability requirements (performance assurance and robustness) of embedded systems. Several examples of partitioned system have already been implemented on a MP211 chip.

IV. Flexibility and Scalability

A. Shift to Logical Partitioning

Partitioning by physical cores can achieve reliable systems as we mentioned, to be sure, but when we adapt it to wider range of target devices, it may become a subject of discussion:

- Flexibility
  Close correspondence between each system model and the number of physical cores could be considered as a sore spot. It is undesirable to design and manufacture a custom version of multicore LSIs for each partition model of the embedded systems, since manufacturing many kinds of LSIs in small quantities will become economically prohibitive in the near future, due to the rising costs of design and verification of complicated logic, and the increasing cost of LSI masks and manufacturing fabs.
- Scalability
  Performance assurance and robustness in the physical partitioning approach mentioned depend on the hardware architecture of AMP. AMP is not necessarily the best architecture for performance acceleration of a single application, where SMP (Symmetric Multiprocessing) may be preferable.

Towards more flexible and scalable architecture for the future embedded systems, we think it necessary to shift from physical partitioning to something more flexible; logical partitioning. Logical partitioning is an approach for fulfilling the required performance characteristics on a single hardware architecture, utilizing the software mechanisms of resource partitioning and scheduling. Mere denial of physical partitioning, however, leads to the disadvantages of software approach mentioned in section II, and we have to seek better combination in the tradeoffs. In the following, we show two topics which deal with the tradeoffs.

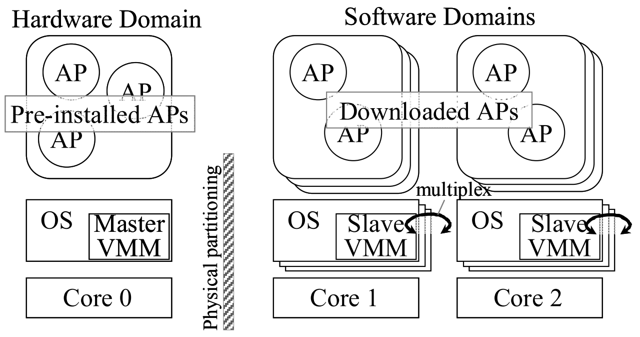


Fig. 7. VIRTUS architecture

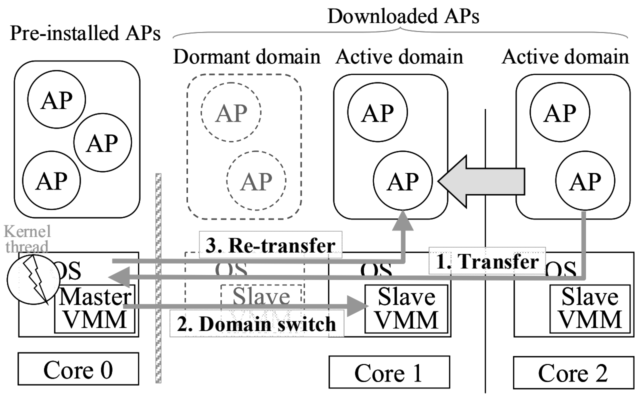


Fig. 8. Asymmetric VMM and domain switch mechanism in VIRTUS

B. Flexibility by Processor Virtualization

Since the FIDES platform mentioned relies upon partitioning by physical cores, system-wide application management would become complicated if the number of required domains exceeded the number of cores in a real multicore LSI. We developed a dedicated processor virtualization architecture called "VIRTUS" to fill the gap [5].

VIRTUS aims to provide an arbitrary number of secure and efficient execution domains. Domains partitioned by hardware (physical partitioning such as FIDES) are good in execution performance but their number is limited to the number of cores in the chip. Domains created by software such as virtual machine middleware are flexible and low cost, but their overhead reduces system performance significantly. In the tradeoff between them, VIRTUS uses the combination of both; a base domain, in which confirmed reliability and good execution performance are necessary, is based upon a hardware domain, and other domains with less reliability expected, such as downloaded application domains, are implemented on other cores using software domain multiplex mechanisms (Fig. 7). Note that the base domain is surely protected from any unwanted interference from other domains because they are physically partitioned.

The core of VIRTUS is a unique virtualization scheme, an asymmetric virtual machine monitor (AVMM), shown in Fig. 8. A set of AVMM is composed of one master VMM on the base domain and one or more slave VMMs on other domains. The master VMM, which is very reliable due to its location (in

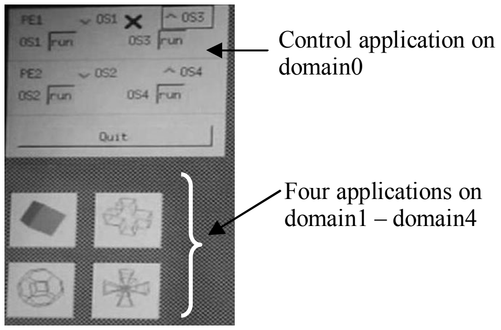

PE1
OS1
OS3
OS1 run
OS3 run
PE2
OS2
OS4
OS2 run
OS4 run
Quit
Control application on domain0
Four applications on domain1 – domain4

Fig. 9. VIRTUS demonstration – 5 domains on 3 cores

the base domain), handles context switches of software domains and manages inter-domain communications. When many software domains are multiplexed and a task in a running domain initiates communication to a task in a dormant domain, the communication messages are routed via the master VMM, the master VMM switches the domain context, and finally the messages are received by the destination domain.

Thus VIRTUS can effectively execute any number of OS instances that exceed the number of physical CPU cores, enabling flexible design of embedded systems with an arbitrary number of domains. Fig. 9 shows a screenshot where five OS instances are created on three CPU cores in our MP211 chip.

C. Scalability by SMP Based Hardware

In contrast with software side flexibility by VIRTUS above, hardware side flexibility is enhanced by an architectural shift from AMP to SMP. Combined with an SMP OS, SMP hardware could execute large-scale applications that span several cores, with higher efficiency than that for AMP. ARM and we jointly developed the world's first embedded SMP processor, MPCore [6]. Its first test chip has integrated four ARM11 cores with cache coherency mechanisms, and SMP Linux ported to MPCore exploits a scalable performance boost by multithreading and multitasking.

For embedded systems, however, the SMP model leaves room for improvement. The characteristics of a pure SMP OS, which primarily aims at higher throughput, do not fit performance assurance nor secure domains; an application program on an SMP OS could easily exhaust most of the CPU time and/or memory bandwidth, and a virus could damage not only the infected application but also all the tasks on the SMP OS altogether. One of the known approaches to this problem is to assign specified tasks (e.g. realtime processing tasks) definitely to a specific CPU core, using the processor affinity function [7]. Another one is an AMP/SMP hybrid approach, where the single SMP OS kernel applies AMP scheduling to some tasks specified [8]. These methods aim to balance reliability with the flexibility of multicore systems by replacing some of the SMP features with AMP ones. We believe that proper control of schedulers that assign hardware resources to software tasks is the key to the reliability in the future embedded systems.

V. Summary

Multicore architecture can be used to improve system reliability, which is of primary importance in embedded devices. The basic concept is to suppress unnecessary interference by functional partitioning. Partitioning by physical cores is considered the most reliable, and has been proven feasible on the specific multicore LSI MP211, with our OS wrapper mechanism and our dynamic access control scheme to relax the communication restrictions caused by the strict partitioning. In order to adapt the approach to wider range of embedded devices, we need to shift to logical partitioning which has more flexibility and more scalability. We plan to incorporate the concepts of virtualization and SMP into the strict physical partitioning approach, so that we should reach the best point in the tradeoffs for the future reliable embedded systems.

Acknowledgements

The authors thank all the MP98 project members for their cooperation and support.

References

[1] D. Geer, "Chip Makers Turn to Multicore Processors," IEEE Computer, pp11-13, May 2005.

[2] J. Sakai, et al., "Multi-Tasking Parallel Method on MP211 Multi-core Application Processor," in Proceedings of the IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips VIII), pp198-211, April 2005.

[3] S. Torii, et al., "A 600MIPS 120mW 70uA Leakage Triple-CPU Mobile Application Processor Chip," in Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC), Digest of Technical Papers, pp136-137, February 2005.

[4] INOUE, H, A. Ikeno, M. Kondo, J. Sakai and M. Edahiro, "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 (CODES+ISSS) , pp178-183, September 2005.

[5] INOUE, H, A. Ikeno, M. Kondo, J. Sakai and M. Edahiro, "VIRTUS: A New Processor Virtualization Architecture for Security-Oriented Next-Generation Mobile Terminals," in Proceedings of the 43rd Design Automation Conference (DAC), pp484 - 489, July 2006.

[6] J. Goodacre, A.N.Sloss, "Parallelism and the ARM instruction set architecture," IEEE Computer, pp42-50, July 2005.

[7] S. Brosky and S. Rotolo, "Shielded processors: guaranteeing sub-millisecond response in standard Linux," in Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03), April 2003.

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