Executing secured virtual machines within a manycore architecture

Clément Dévigne, Jean-Baptiste Bréjon, Quentin L. Meunier, Franck Wajsbürt

Executing Secured Virtual Machines within a Manycore Architecture

Clément Dévigne, Jean-Baptiste Bréjon, Quentin Meunier and Franck Wajsbürt
Sorbonne Universités UPMC Univ Paris 06, CNRS, LIP6 UMR 7606 4 place Jussieu 75005 Paris
Email: {clement.devigne,jean-baptiste.brejon,quentin.meunier,franck.wajsburt}@lip6.fr

***Abstract*—Manycore processors are a way to face the always growing demand in digital data processing. However, by putting closer distinct and possibly private data, they open security breaches.**

This article presents undergoing work aiming at providing security guaranties to different users utilizing different cores in a manycore architecture. The proposed solution is using physical isolation and a hypervisor with minimum rights, although the work described in the paper focuses only on hardware mechanisms. We present a hardware module providing an address translation service allowing to fully virtualize operating systems, while offering advantages compared to a classical memory management unit within our context. Experiments made on a virtual prototype shows that our solution has a low time overhead – typically 3% on average.

I. Introduction

The computer world is facing an explosion in the amount of digital data. This data can come from social networks as well as new uses of mobile computing as communicating objects. The information contained in this data is valuable either for commercial purpose, or for economic, environmental or health-related purposes as well. Clearly, the issue of security for accessing such information and the protection of personal data are critical.

By their nature, manycore processors are able to run multiple applications in parallel and thus allow to process a large data stream. However, they must be able to guarantee the security properties for such applications, in particular integrity and confidentiality.

The TSUNAMY ANR project [1] aims at proposing a mixed hardware/software solution allowing to execute numerous independant applications, while providing an isolated execution environment as a response to the confidentiality and integrity problematics. The choice of a manycore architecture seems particularly suited to this goal, and the challenges of the project are to render such an architecture secured. The baseline manycore architecture used in this project is the TSAR [2] architecture, which is a manycore architecture with hardware cache coherence and virtual memory support, but no particular mechanism for addressing security issues. Our modified targeted architecture will be called the Tsunamy architecture.

The proposed architecture can typically be used by servers (e.g. in the cloud), to which several clients with different needs can connect and execute their program for processing data. In such a context, it makes sense that clients are isolated with more than just the process notion, because a bug exploit in the Operating System (OS) could lead to data leakage and corruption between two processes run by two different clients. In our proposed solution, we make thus the assumption that each client runs an entire OS.

In this article, the term *virtual machine* refers to an OS running on a subset of the architecture, and isolated from other hardware and software elements. The virtual machines must be protected against: (1) unauthorized reads of data (confidentiality), (2) unauthorized modifications of pieces of data read or transmitted by the VM (integrity) and (3) information leakage – data left in memory or hardware components which can be exploited by another malicious virtual machine.

If our proposed solution mixes hardware and software, this article focuses on the hardware part. Although the work presented in this article is still in progress, we believe that this paper still makes two contributions:

- we propose a light hardware design implementing virtual machine isolation;
- we demonstrate the feasibility of our solution by the achievement of a cycle-accurate prototype of the proposed architecture.

The rest of the document is organized as follows: section II gives more details about the context and discusses related works; section III contains a description of the existing components upon which this work is based; section IV details the proposed hardware components; section V presents preliminary simulations results; finally, section VI concludes and summarizes the remaining work.

II. Related Work and Hypothesis

Operating system virtualization [3] is a technique which allows to execute an unmodified OS on a part of an architecture. An hypervisor is generally used to manage the different virtualized operating systems [4], [5].

In this work, we also aim at integrating an hypervisor, but with as few rights as possible, in order to reduce the effects of a possible bug exploit in it. The hypervisor is a trusted software agent which manages the resource allocation and thus it is in charge of starting and stopping virtual machines. However, it does not have the rights to access hardware components allocated to a virtual machine until its destruction, in particular L2 caches. The hypervisor itself runs on dedicated cores, and has a dedicated L2 cache – in the following, we make the hypothesis that a cluster is dedicated to the hypervisor.

Besides, to enforce isolation between the different virtual machines, we use physical isolation, guaranteed by the hardware. The price to pay is a lack of flexibility in hardware resource allocations: a user launching a virtual machine cannot allocate less than a cluster, and clusters can be physically isolated by the means of address routing.

Admittedly, this hardware has to be configured by the hypervisor. However, in our final solution, the hypervisor won't be able to access data inside clusters, nor change the

configuration once the primary configuration is made. Indeed, we intend to make the reconfiguration possible only after completion of a hardware procedure comprising the deletion of all memory banks contents.

The technique employed to physically isolate the concurrently running virtual machines on the architecture is to add a third address space, whose addresses are called *machine addresses*. The translation mechanism ensures that all the machine adresses obtained for a virtual machine can only address targets located inside the cluster allocated to that virtual machine (with the exception of some peripherals, which will be discussed later). Traditionally this translation is made *via* the MMU inside the first-level cache [6], [7], [8] but it requires that the hypervisor and the virtual machine share cores. In our case we do not want that virtual machines share core or memory bank resources with the hyperviseur (or other virtual machines). Our translation mechanism operates at the output of the first-level cache, before the intra-cluster crossbar and is performed by the means of a hardware component called Hardware Address Translator (HAT). This module acts as a wrapper for initiators inside a cluster and plays the same role as a MMU, but differs in several ways compared to the latter.

First, a MMU generally uses a translation cache (called TLB) to speed up address translation. This implies a non negligible hardware overhead, including the logic to manage the TLB misses, while HATs only need topology information to perform address translation. Second, the hypervisor must create the page table for the memory allocated to a virtual machine and store it into a memory space non accessible by itself nor any virtual machine. This cannot be done entirely in software and therefore also requires the introduction of a specific hardware element. Third, a MMU is necessarily slower to perform address translation because of the TLB misses overhead. Finally, the main advantage of a MMU is that it translates with a page granularity (e.g. 4KB) while a HAT operates with a coarser granularity (cluster granularity). The page granularity can be useful when virtual machines share memory banks, but this is not within our hypothesis to physically isolate the virtual machines.

In summary, the HAT is a light and fast component compared to a MMU, whose downside is the translation granularity, but that does not comes as a problem with our hypothesis.

III. Existing Hardware Components

A. The TSAR Architecture

Figure 1 shows an overview of the TSAR architecture. It is a clustered architecture with a 2D mesh topology using a Network-on-Chip (NoC). The cluster with coordinates $(0, 0)$ additionnally contains an access to I/O peripherals, and is called I/O cluster. The TSAR architecture is design to support up to 16×16 clusters.

All clusters contain:

- 4 MIPS cores with their paginated virtual MMU and their first level caches, split between instructions and data. The L1 cache coherence is managed entirely in hardware. Misses in the Translation Lookaside Buffer (TLB) are also handled by the hardware.
- 1 second level (L2) cache, which is in charge of a segment of the physical memory address space. In particular, it is responsible for the coherence of the copies in L1 caches for the lines contained its segment.
- 2 internal peripherals: an interrupt controller including timer functions (XICU) and a DMA controller. These peripherals are called replicated peripherals.
- A local crossbar interconnecting these components with an access to the global network *via* a router.

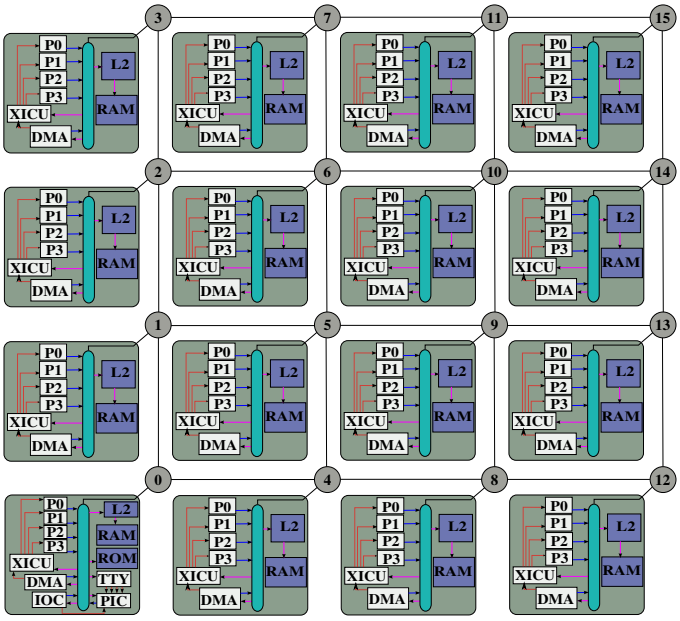

Figure 1. The TSAR Architecture with 64 Cores Spread over 16 Clusters

The I/O cluster additionally contains:

- A terminal controller (TTY).
- A hard-drive disk controller (IOC).
- A Programmable Interrupt Controller (PIC), able to convert a hardware interrupt into a software one.

This architecture will be used as a base for our secured architecture proposal, with substantial modifications in order to meet the motivated requirements.

IV. Hardware Modifications to the TSAR Architecture

This section presents in details the hardware modifications proposed by our solution in order to isolate two virtual machines by the means of hardware.

The translation from physical addresses to machine addresses is performed by the HAT module, which is configured once by the hypervisor at the start of an OS and placed behind each initiator in the architecture – as well cores as replicated DMAs.

A machine address outgoing from an HAT can be one of the followings two types:

- an address targetting a module included in a cluster of the same virtual machine: memory *via* the L2 cache or a replicated peripheral (DMA or XICU); this is the stardard case, and it will be refered to as an *internal access*
- an address targetting a peripheral outside the virtual machine, namely the disk controller or the TTY This case will be refered to as an *external access*.

A. Internal Accesses

The machine address space is split on the clusters in such a way that the most significant bits (MSB) define the cluster coordinates, as this is the case with the TSAR architecture (except that addresses are called physical). Thus, the translation from the physical addresses to machine addresses only consists in changing the MSB to change the cluster coordinates: this represents at most 4 bits for 16 clusters. Figure 2 illustrates how the HAT module works with a four clusters architecture.

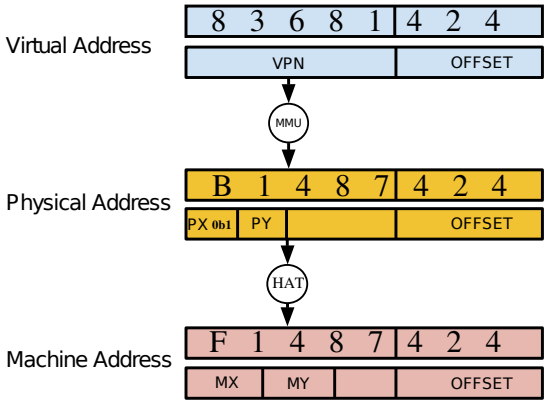

Figure 2. Translation from Physical Address to Machine Address for an Internal Access

The address is sent by a processor in a virtual machine running on clusters 1 and 3 of the platform. The X size of the virtual machine is 2 and the Y size is 1. The *Pxl* field represents the number of bits needed to code the X size of the virtual machine (in our case just 1 bit is needed). The *Pyl* field is the same as *Pxl* for Y size – in our case, 0 bit.

A processor sends the virtual address 0x83681424 which is translated by the MMU into the physical address 0xB1487424. In a virtual machine with 2 clusters, the physical address starting with 0xB is located in the second cluster, containing the physical address range <0x80000000–0xFFFFFFFF>. In our example the second cluster of the virtual machine is the fourth cluster of the platform, with a machine address range <0xC0000000–0xFFFFFFFF>. Therefore, the HAT will provide the translation of an address belonging to the second cluster of the virtual machine to the fourth cluster of the platform.

B. External Accesses

The hypervisor software is not involved in the accesses made by virtual machines to peripheral devices. The differentiation between an internal access and an external access is made *via* a bit in the physical address. This bit, called the *DEV* bit, is the one just after the (x,y) coordinates bits. If the *DEV* bit is 1 the HAT acts similarly as a segmentation mechanism [9], and uses a table to check if the request is valid. This is the case if the device targeted by the address is actually allocated to the virtual machine. This device access table contains several entries, each one containing two pieces of information:

- the base physical address of the segment associated to the device;
- the two's complement of the size in bytes of this segment.

For simplicity reasons, the IO cluster is chosen as the cluster running the hypervisor, though it could be any other cluster.

Figure 3 provides an example of the way the HAT performs a device access.

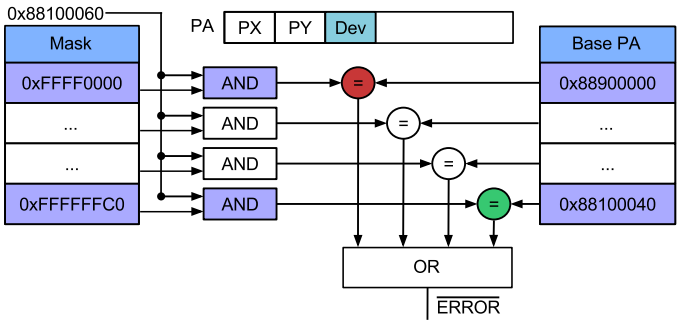

Figure 3. Translation from Physical Address to Machine Address for an External Access

The virtual machine of our example is deployed on 4 clusters, so the fields *px* and *py* are 2-bit wide. The physical address 0x88100060 emitted by the processor has therefore the bit *DEV* set. This address is then masked with the two's complement contained in the table and compared with the base physical addresses of all devices associated to the virtual machine. If one comparison is true then the physical address is valid and the request is sent to the target device. In contrast, if no comparison is true then an error is raised and returned to the processor (as a bus error) issuing the request; this error indicates to the operating system that the processor tried to access a non-existing address.

C. Peripherals Allocation

The external access mechanism provided by the HAT allows a virtual machine to access specific channels of the IOC and TTY. Each channel contains a set of addressable registers independent from the others. Each channel of the IOC also contains a hard drive disk (or a partition) comprising the image of an OS and its filesystem, and the associated bootloader (refered to as *OS instances*). All the disk images will eventually be cyphered with a key specific to the user, which can in turn be decyphered by a user password and stored somewhere accessible by the hypervisor. Only one OS instance can be run at a time, and in our prototype it is selected in the hypervisor command to launch a new instance.

The association between an instance and the allocated clusters is used by the hypervisor to configure the PIC: the interruptions outgoing from a channel are routed to the PIC, what must trigger software interrupts towards an XICU located in an allocated cluster, which in turn converts it to a hardware interrupt. Eventually, the PIC will only be able to send requests to the XICU peripherals in order to avoid that a corruption in the hypervisor, which configures the PIC, can result in data integrity violation *via* the PIC.

V. Preliminary Results

Currently, we have already been able to run experiments on a Tsunamy platform including 16 clusters and running 4 operating system instances on a various number of clusters.

The architecture is described in SystemC at the cycle-accurate level using the SoCLib components library [10]. OS instances are run *via* an hypervisor terminal, which also allows to switch between the displays of all instances.

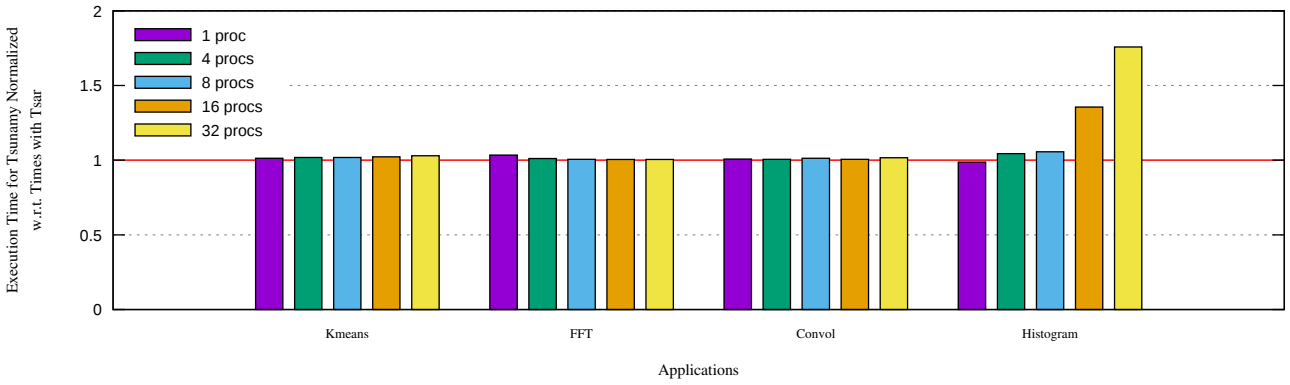

Figure 4. Execution Times for Tsunamy Normalized w.r.t. Times with Tsar

A. Applications

Applications used for evaluations are FFT from the Splash-2 suite [11], Histogram and Kmeans from the Phoenix-2 benchmark suite [12], and Convol, which is an image filtering program performing a 2-dimensional convolution filter. Table I shows the configuration for each application.

Table I. APPLICATIONS PARAMETERS

Application	Input Data
Histogram	25 MB image (3,408 × 2,556)
Convol	1,024 × 1,024 image
FFT	2^{18} Complex Points
Kmeans	10,000 points

All these benchmarks have been run over an operating system called ALMOS [13], which is developed in our laboratory. It is an UNIX-like research OS dedicated to manycore architectures.

B. Results

Figure 4 shows the execution time with Tsunamy for the 4 considered benchmarks. These times are normalized per number of cores and per application w.r.t. times on TSAR.

For Kmeans, FFT and Convol benchmarks, Tsunamy has an average overhead performances of 3%. Results on Histogram can be explained by the nature of the application, in the sense that all the input data come from the hard drive disk and the loading phase is predominant in this application – more than 50% of the execution time. All the virtual machines need to access their disk at the same time. As the disk controller has only one initiator port on the network it becomes a bottleneck during multiple access from several virtual machines.

VI. CONCLUSION

This article presented the hardware part of a mixed hardware/software solution allowing to execute physically isolated virtual machines comprising an unmodified operating system on a manycore architecture. It uses a third address space inducing a light hardware overhead for all initiators, and a very low time overhead – typically 3%.

Undergoing and future work deal with the specification and implementation of the procedure required to stop a running virtual machine.

ACKNOWLEDGMENTS

The work presented in this paper was realized in the frame of the TSUNAMY project number ANR-13-INSE-0002-02 supported by the French Agence Nationale de la Recherche [14].

REFERENCES

[1] LIP6, Lab-STICC, LabHC and CEA-LIST, "Hardware and software managemenT of data SecUrity iN A ManY-core platform," https://www.tsunamy.fr.

[2] LIP6 and BULL, "TSAR (Tera-Scale Architecture)," https://www-soc.lip6.fr/trac/tsar.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[4] C. Dall and J. Nieh, "Kvm/arm: The design and implementation of the linux arm hypervisor," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 2014, pp. 333–348.

[5] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[6] S. Jin, J. Ahn, S. Cha, and J. Huh, "Architectural support for secure virtualization under a vulnerable hypervisor," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 272–283.

[7] S. Jin and J. Huh, "Secure mmu: Architectural support for memory isolation among virtual machines," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, June 2011, pp. 217–222.

[8] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel virtualization technology: Hardware support for efficient processor virtualization." *Intel Technology Journal*, vol. 10, pp. 167 – 177, 2006.

[9] J. Ahn, S. Jin, and J. Huh, "Fast two-level address translation for virtualized systems."

[10] LIP6, "SoCLib : an open platform for virtual prototyping of MP-SoC," http://www.soclib.fr.

[11] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*. New York: ACM Press, 1995, pp. 24–37.

[12] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*. IEEE, 2007, pp. 13–24.

[13] LIP6, "Advanced Locality Management Operating System," http://www.almos.fr.

[14] A. N. de la Recherche, http://www.agence-nationale-recherche.fr.