Dynamic Spatially Isolated Secure Zones for NoC-based Many-core Accelerators

Maria Méndez Real, Philipp Wehner, Vincent Migliore, Vianney Lapotre, Diana Göhringer, Guy Gogniat

▶ **To cite this version:**

Maria Méndez Real, Philipp Wehner, Vincent Migliore, Vianney Lapotre, Diana Göhringer, et al.. Dynamic Spatially Isolated Secure Zones for NoC-based Many-core Accelerators. 8th IEEE International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, Jun 2016, Tallinn, Estonia. 10.1109/ReCoSoC.2016.7533900 . hal-01347175

HAL Id: hal-01347175
https://hal.science/hal-01347175

Submitted on 26 Feb 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Dynamic Spatially Isolated Secure Zones for NoC-based Many-core Accelerators

Maria Méndez Real*, Philipp Wehner†, Vincent Migliore*, Vianney Lapotre*, Diana Göhringer† and Guy Gogniat*
*Univ. Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France, maria.mendez@univ-ubs.fr
†Ruhr-University Bochum, Germany, {philipp.wehner, diana.goehringer}@rub.de

***Abstract*—Many-core architectures are becoming a major execution platform in order to face the increasing number of applications executed in parallel. While many-core accelerator architectures offer users with massive parallelism and high performance, it also introduces some key challenges in terms of security. Indeed, in order to leverage performance, a great number of applications running in parallel may share resources. A malicious application may compromise other applications sharing resources with or the whole system by directly accessing, deducing or retrieving sensitive data. This work focuses on a many-core accelerator architecture extended with mechanisms allowing the logical and spatial isolation of sensitive applications through the dynamic creation of secure zones. Each sensitive application is executed within a secure zone avoiding any resource sharing with other potentially malicious applications, preventing denial of services within the secure zones as well as confidentiality and integrity attacks. A set of services guarantying the dynamic creation and handling of spatially isolated secure zones in a many-core accelerator architecture are proposed. These services are integrated into a software controller on a many-core accelerator architecture and evaluated through virtual prototyping.**

I. Introduction

The increasing number of applications running in parallel has introduced the need of a great parallel computation power. Hardware many-core accelerators offer users with massive parallelism and high performance. In order to leverage performance, applications running in parallel might share resources such as communication infrastructure, computation and memory resources introducing some key challenges in terms of security. Indeed, a malicious application may compromise other applications sharing resources with or even the whole system by illegally accessing the memory, deducing or retrieving sensitive data or preventing other applications from using the available resources. This work focuses on the secure deployment of sensitive applications on hardware many-core accelerators and proposes an approach in order to dynamically create and handle spatially isolated secure zones for the execution of sensitive applications. In this approach, a sensitive application is executed in an isolated secured zone in order to avoid the vulnerabilities caused by sharing resources. A hardware NoC-based many-core accelerator offering a great number of resources is considered. A software processor on the accelerator acts as a controller of the platform and is charged with the secure and dynamic applications deployment and management. For this purpose, a set of services executed on the accelerator controller are proposed. Different deployment strategies are studied and compared on different execution scenarios through virtual prototyping. The main contributions of this work are

- a set of new secure-enable mechanisms for a many-core accelerator controller, able to dynamically create secure zones in order to logically and spatially isolate sensitive applications avoiding cache Side-Channel Attaks (SCA) and Denial of Services (DoS) within the secure zones,
- the validation and comparison of different deployment strategies and the evaluation of the performance overhead induced through virtual prototyping.

The remainder of the paper is organized as follows: Section II presents the considered system, the associated threat model and the proposed secure-enable approach. In Section III, related work on similar and extended problematics is depicted and compared to the aim of this work. Section IV explains the principle and integration of the proposed mechanisms. In Section V and VI, the evaluation environment and the evaluation of the proposed approach are respectively presented. Finally, Section VII draws conclusions and future work.

II. Problem statement and proposed approach

This work focuses on the execution of parallel applications on many-core accelerators and proposes an approach in order to securely deploy and execute sensitive applications.

A. Many-core accelerator context

A many-core accelerator offering hundreds of computing resources that can be heterogeneous is considered. A system running an Operating System (OS) might execute a great number of parallel applications. The OS launches the applications and can delegate part of or the entire computation workload to the many-core accelerator which offers the needed resources. When launching an application, the user specifies if the application needs to be executed spatially isolated within a secure zone. The many-core accelerator is composed of clusters connected through a 2-Mesh NoC. Each cluster encompasses some computing and memory resources. Each processor is directly connected to its local memory containing the code, heap and stack via a local bus. Processors within a cluster can access as well some shared resources such as shared memory and peripherals through a shared bus. Figure 1 shows an overview of the considered system. Some examples of such clusterized NoC-based architectures are [1] and [2].

B. Threat model

For this study, the hardware platform as well as the services executed on the controller are considered trusted. Three main kind of attacks can then be considered: Confidentiality and Integrity (C&I) and Denial of Services (DoS).

C&I: C&I attacks refer to the illegal access, directly or indirectly, to the data by reading or writing. This can be achieved by directly accessing to an illegal partition of the memory by reading or writing the memory, or indirectly by collecting some information during the execution of sensitive applications and deducing some more important information. For instance, a malicious application sharing cache memory with a sensitive application can launch Cache Side Channel Attacks (Cache SCA) and deduce some sensitive data by analyzing

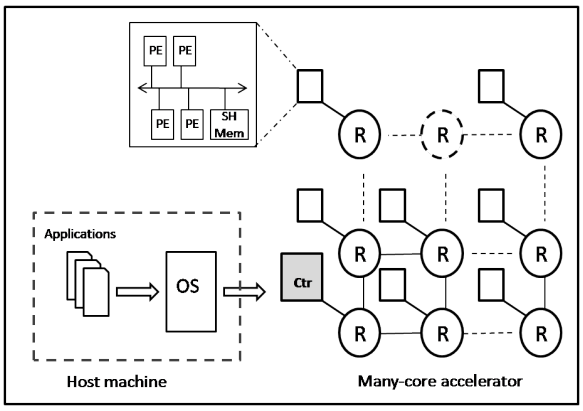

Fig. 1. Overview of the considered system

the access patters to the cache of the sensitive application [3] [4]. Furthermore, leakage of information allows the attacker to deduce potentially sensitive data as well, accessing to long-lasting data for instance. Moreover, communication SCA allow to deduce some information when applications share the communication infrastructure.

DoS: These attacks aim at preventing other applications from using the system resources in order to low their performance. These kind of attacks can also try to block the system. A compromised application can for instance launch DoS attacks by requesting a large number of resources (memory or computing) e.g. creating a large number of threads, with the objective of saturating the system or decreasing the performance of other applications needing these resources. Malicious applications can work together as well in order to tamper with the system.

C. Logical and spatial isolation

Applications are vulnerable when sharing resources. In order to guarantee a secure deployment of applications on a many-core accelerator avoiding cache SCA, we propose a set of services executed by the accelerator controller in order to perform a logical and spatial isolated execution of sensitive applications. Logical isolation refers to the mediation of direct memory access in order to avoid direct illegal access to the memory. This can be achieved for example through a Memory Management Unit (MMU). Logical isolation is not sufficient since indirect illegal access to data (e.g. Subsection II-B) nor DoS attacks are avoided. The solution proposed in this paper is achieved by dynamically creating a spatially isolated secure zone for the deployment of a sensitive application. A secure zone is composed of a number of contiguous clusters, in which all the resources are dedicated to one single sensitive application. In consequence, this application will not share its computating and memory resources preventing cache SCA. Once the application is finished, the secure zone is dissolved and the resources are released and reset in order to avoid any leakage of information. While communication SCA within the secure zone clusters are avoided, the remote access outside the secure zone are not protected. There have been some efforts on the protection of communication through the NoC [5]. Such solutions can be adapted and integrated in our system in order to protect communications outside the secure zones (this point will be considered in future work). overhead are expected. Two different deployment strategies are considered:

Static secure zone size: One possible approach is to create a secure zone composed of a certain fixed number of clusters that will be dedicated the entire execution time of the isolated application.

Dynamic secure zone size: Another approach in order to minimize the under-utilization of resources within a secure zone is to dynamically adapt its size. In this approach, clusters can be dynamically added or released according to the needs of the isolated application.

III. Related work

There is little work that addresses the problem of secure dynamic deployment of parallel applications on many-core architectures.

Platform bi-partition and MMU/MPU. ARM TrustZone [6] provides hardware support for the creation of Trusted Execution Environments (TEEs) and therefore the isolation of applications within the same processor. However, at any time instant, only a single domain in the system can be secured. Therefore, the isolation is only achieved at the processor level. The use of MMU and Memory Protection Unit (MPU) allows the secure partition of shared memory mediating memory access in order to avoid confidentiality and Integrity attacks (C&I). Bi-partitioning, nor MMU, MPU are not enough anymore. Indeed, applications running on different processors are not protected from each other since sharing the communication infrastructure leads to possible leakage of information attacks such as SCA.

Data protection in NoC based multi-core architectures. In [7] [8], authors present a ”NoC MPU” for shared memory NoC based multi-core architectures in order to isolate the shared memory partitions. In the architecture proposed, each NoC node is encompassed by whether an initiator or a target device. NoC MPUs are located at the initiator side network interfaces. The partition access rights tables are configured by the OS. This solution avoids C&I attacks, however, DoS and SCA remain possible.

Security capabilities in modern multi-core and many-core architectures. In [9], authors propose ”illegal access probe” and ”denial of services probe”. These solutions tackle illegal direct access to the memory as well as DoS attacks. In [10], authors explore the security opportunities enabled by existing many-core systems. They propose the extension of the Intel Single-chip Cloud Computer many-core platform with security properties such as isolation running a trusted agent (Trusting Computing Base (TCB) element) on every core of a many-core platform. Moreover, a new security property called application awareness in order to allow each application to protect itself from a compromised kernel is introduced. This new property is interesting, nevertheless, no actual implementation has been proposed.

Compared to these efforts, this work focuses on the isolated execution of parallel applications on a standalone many-core accelerator and proposes a set of services for a system controller able to dynamically create physical isolated secure zones, similar to TEEs but running in parallel in many-core platforms in order to avoid DoS and C&I attacks.

IV. Principle and integration of the proposed secure-enable mechanisms

Several services executed by the accelerator controller are proposed in order to guarantee the isolated and isolated execution of sensitive applications within secure zones.

Monitoring the status of the platform. In order to create and handle secure zones, monitoring mechanisms providing information

about the global state of the physical platform and available resources in terms of physical memory, active tasks and processor utilization rate are implemented. The accelerator controller controls the deployment of applications and thus is aware of which tasks are active and which processors are used. The controller services are executed on a specific processor in a specific and known cluster. In order to inform the controller when a task is finished, each task, before the end of its execution sends a message to a specific address of the memory of the cluster encompassing the controller processor. This latter, periodically accesses to its cluster memory in order to update the platform state structures. Monitoring services are consulted by the controller each time a resource allocation decision needs to be taken.

Non-isolated application mapping. Memory within cluster is statically partitioned into the number of processors encompassing the cluster (4 in our case). For each running task of an application, one partition is allocated to it when the task is mapped. When a new non-isolated application needs to be mapped, the controller consults the platform state structures in order to find an idle processor. If this latter exists, then the first task of the application is mapped, the state structures are updated and the corresponding memory partition is allocated.

New task mapping. The controller structures keep track of the processor allocated to each task. A task might need to create other tasks. In order to leverage performance, when a new task needs to be mapped, the mapping algorithm takes into account the location of the father task, and tries to leverage the locality of memory accesses by mapping the child task as close as possible. If no idle processor is found, then the controller will try to map this task the next scheduling tick and the task will wait until a resource is available.

Creation and release of secure zones. Two different approaches are considered concerning the size of the secure zones.

Fixed size secure zone: Here the controller aims at finding a specific number of contiguous idle clusters before it can map a new isolated secure zone. The algorithm searches for contiguous idle clusters in order to build a secure zone in which all the resources will be dedicated to the sensitive application during all its execution time. Notice that once the isolated application is finished, the resources are released and the memory within the dedicated clusters is reset in order to avoid any leakage. The tasks created by a task executed in a secure zone will be mapped within the secure zone. If there is no idle resources within the secure zone, the new task will wait the release of a processor within its secure zone. Algo. 1 presents the algorithm charged with the creation of a fixed size secure zone. While the number of contiguous clusters passed into parameter has not been reached, and all the idle clusters have not been considered, the algorithm takes an idle cluster and adds it into a list of clusters. Each time a cluster is added, the algorithm searches into its 4 potential contiguous clusters an idle one and adds it into the list. If all the clusters have been considered and no idle contiguous clusters enough have been found, the algorithm fails on creating a secure zone, and it will try again at the next scheduling tick.

Dynamic size secure zone: In order to minimize the under-utilization of resources within secure zones, a second approach in which the size of the secure zone is dynamically adapted is also considered. In this approach, an isolated application needs only one single idle cluster in order to start to be executed. When the application needs more resources, the controller searches for a contiguous idle cluster. If this latter exists, then this one is added to the secure zone, tagged as dedicated and the platform state is updated.

Algorithm 1 Creating a fixed size secure zone

```
1: Input: the architecture A, a number of contiguous clusters required by a sensitive
   application NbR
2: Output: a set of contiguous clusters forming a secure zone SZ[NbR]
3: i=0
4: while i < NbR OR all the clusters ∈ A are crossed do
5:     for each idle cluster C_j ∈ A do
6:         SZ[i] ← C_j
7:         if i < NbR then
8:             while i < NbR AND all its potential neighbor clusters have not been
               considered do
9:                 if i < NbR and this neighbor cluster C_k is idle then
10:                    i ++
11:                    SZ[i] ← C_k
12:                end if
13:            end while
14:        end if
15:        if i < NbR then
16:            empty the SZ and restart from another idle cluster
17:        end if
18:    end for
19: end while
```

On the other hand, if no additional cluster can be added to the secure zone, the isolated application may need to be sequentialized and the controller will try to extend the secure zone again at the next scheduling tick. In the same manner, if a cluster within a secure zone is not used anymore, this one is released decreasing the size of the secure zone.

Back to the threat model. The proposed approach guarantees that the resources within clusters of a secure zone are dedicated to a single isolated application avoiding any resource sharing with other applications within these clusters. Hence, cache SCA, previously possible through the local bus within secure zone clusters, are prevented. Moreover, a malicious application can not prevent the isolated application from using the dedicated computing and memory resources within the secure zone clusters launching DoS attacks. However, the NoC is not protected. In fact, DoS attacks on the NoC as well as cache SCA through the NoC remain possible. Solutions in order to protect the communication through the NoC, such as [5] can be adapted and integrated. This will be considered in future work.

The proposed services have been integrated and evaluated in terms of induced performance overhead and resource utilization rate through a virtual prototyping environment.

V. Evaluation environment

MPSoCSim: MPSoCSim [12] is an OVP-based simulator, allowing in its extended version, the evaluation of distributed NoC-based multi-core and many-core architectures [13]. This latter relies on a system level modeling language SystemC NoC where each router is connected to a SystemC TLM Network Interface (NI) connected to a local group called cluster. Each cluster can encompass up to 4 subgroups and shared resources between subgroups such as a shared memory and a NI to connect the cluster to the NoC. Figure 2 shows the structure of each cluster. Each subgroup is composed of one processor directly connected as a master to its local memory through a local bus. The local memory stores heap, stack and the processor code. Each processor can read and write to its own local memory and to the shared RAM within its cluster. The shared RAM is used to exchange data between processors within the same cluster as well as with distant clusters by message passing through the NoC. A shared bus connects subgroups and the shared resources (shared RAM, NI, peripherals,...) to the NoC through the corresponding router.

MPSoCSim validation and HW implementation comparison:

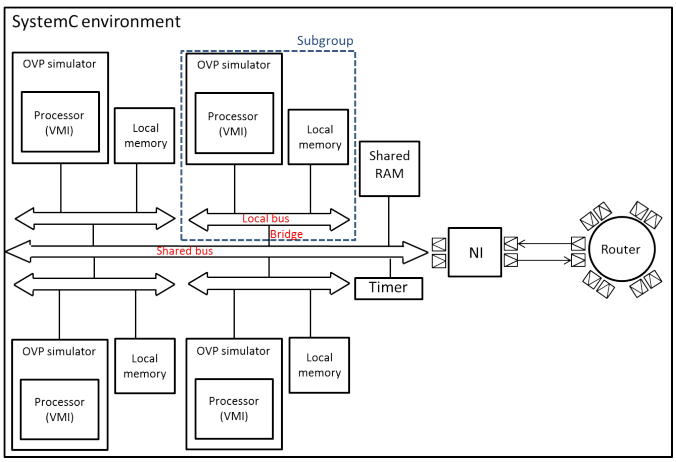

Fig. 2. MPSoCSim simulator in its latest version (1 cluster)

Parameters	Chosen value
Quantum period	10us
Cortex A9 ARM Frequency	667MHz
MicroBlaze (MB) Frequency	100MHz
Nominal MIPS	100
Real flit time (ARM)	850ns
Real flit time (MB)	40ns
Clock delay pass through	1
Network frequency	100MHz

TABLE I
SYSTEM PARAMETERS USED FOR EVALUATIONS

The original version of MPSoCSim has been validated through the comparison with the HW implementation on a Xilinx Zynq device [14]. Table I sums up the used system parameters. Besides the size of the NoC and the number of subgroups in each cluster, the same parameters have been used for the evaluation of the security-aware mechanisms presented in this paper (SectionVI-B). Results in [12] show a deviation of the execution between 2.5% and 17% compared to the HW implementation for the evaluated scenarios.

VI. SECURE-ENABLE MECHANISMS EVALUATION

In this section, the experimental setup is presented and the evaluation of different deployment strategies is discussed.

A. Experimental setup

Many-core accelerator environment: The proposed deployment strategies have been evaluated and compared through virtual prototyping using the MPSoCSim environment introduced in Section V. A 4×4 NoC accelerator is considered. One cluster encompasses only one ARM which acts as the many-core accelerator controller. The rest of the clusters encompass 4 MicroBlazes (MBs) (60 MBs. and 1 ARM in total). Table I sums up the system parameters used to validate and compare the different deployment strategies evaluated in this work.

Many-core accelerator controller: For these experimentations, the ARM acts as a controller of the many-core accelerator. It performs the decision algorithms, described in Section IV, able to dynamically deploy the applications and to dynamically create and handle the secure zones.

Applications: Synthetic applications with a high degree of parallelism such as matrix multiplications are used. Applications encompassing 17 tasks running in parallel are considered. When launching an application, the user might specify if this latter requires to be isolated executed in a secure zone. For each matrix multiplication, the first task called the ”master” task generates the matrices, splits the computation and dynamically sends the data to the potentially distant clusters running the other ”slave” tasks. These latter access to the shared memory within their cluster, perform the corresponding computation and send back to the master task the generated results. The master task collects all the results. Each task, sends a message through the NoC when the task is finished in order to inform the accelerator controller. This latter can thus release the corresponding processor and flush the corresponding memory partition. Applications are duplicated in order to increase the workload on the platform. All the applications are supposed to be ready to execute from the beginning of the execution scenario. However, a priority level is associated to each application in order to determine the applications scheduling order (a Round Robin scheduling with priority is used).

Evaluated deployment strategies: Different deployment strategies for the creation and handling of the secure zones have been considered:

- **Strategy 1.** Secure zones with a fixed size encompassing all the resources needed by the spatially isolated application. In our case this corresponds to 5 clusters (blue bars in the histograms Fig. 3 and 4).
- **Strategy 2.** Secure zones with a fixed size restrained to 4 clusters (red bars in the histograms Fig. 3 and 4).
- **Strategy 3.** Secure zones with a size dynamically adapted (beige bars in the histograms Fig. 3 and 4).

Evaluated execution scenarios: For these experimentations 5 applications, each encompassing 17 tasks in parallel, meaning 85 tasks in parallel in total are sufficient in order to achieve 100% of resource utilization rate during enough time. Besides the baseline scenario with any secure-enable mechanisms, different execution scenarios have been evaluated:

- **Execution scenario a.** 1 of 5 applications is spatially isolated. This one has the the highest scheduling priority.
- **Execution scenario b.** 1 of 5 applications is spatially isolated. This one has a priority 4 (Priority level 1 being the highest one and 5 the lowest one). This allows taking into account the load of the platform when creating and handling a secure zone.
- **Execution scenario c.** 3 of 5 applications (51 tasks of 85) are spatially isolated. In this scenario, applications requiring to be isolated have priority levels 1, 3 and 5 and non-isolated applications 2 and 4 respectively. This allows balancing the scheduling priority level of isolated and non-isolated applications.

Each of these different execution scenarios (a, b and c) will be studied and compared for each proposed deployment strategy (strategies 1, 2 and 3). The evaluated pair will be noted (strategy.execution scenario).

Evaluated results: The different execution scenarios are evaluated for each proposed deployment strategy in terms of

- total application execution time (encompassing the controller deployment services and the execution time of every isolated and non-isolated application),
- execution time in average for isolated and non-isolated applications,

- total time spent on the services executed on the accelerator controller directly impacted by the creation and handling of secure zones and
- computing resources utilization rate.

B. Evaluation results

Total execution time: Fig. 3 shows the total execution time for each (strategy.execution) pair normalized by the baseline scenario where any application is isolated. Results show that for the first execution scenario (a) where the isolated application has the highest priority, the execution time is not impacted by the deployment strategy. This is because there is no load on the platform when the secure zone is created. On the contrary, when there are active tasks on the platform, finding idle contiguous clusters in order to form a secure zone becomes more difficult (scenarios b and c). Moreover, experimental results show that the performance overhead induced by the proposed secure-enable mechanisms increases with the number of required secure zones but remains below 31% of the baseline execution time (Fig. 3). The total execution time when the secure zone is restrained to 4 clusters is greater than for 5 clusters. This is because when 5 clusters are dedicated to a secure zone, the isolated execution achieves its maximum parallelism and the resources are released sooner than when the secure zone is limited to 4 clusters (5 clusters dedicated during 40% instead of 4 clusters dedicated during 60% of the total execution time). Indeed, in this second case (2.b), the isolated execution needs to be sequentialized. Its execution time (and thus the total execution time) is then greater. Moreover, since at the end, only two processors are active (the processors executing the master task and the last task that waited for resources), the utilization rate of the dedicated resources during the execution time of the isolated application is lower than for the scenario a (65% versus 85% Table III).

Execution time in average for non-isolated and isolated applications: Table II shows the impact of each strategy on the isolated and non-isolated applications. It can be seen that the first execution scenario leverages the performance of the isolated applications and penalizes the non-isolated applications (up to +29% of the non-isolated applications execution time in average). This is explained because the secure zone encompasses all the needed resources. While the strategy number 2 seems to penalize both, the dynamic secure zone size minimizes the overhead introduced for the non-isolated applications. This is because the resources are sooner and dynamically released and thus the non-isolated applications can sooner use them again.

Total time spent on the controller services directly impacted by the creation and handling of secure zones: Time spent on monitoring mechanisms is not considered since these have not being modified nor impacted by the proposed secure-enable mechanisms. Fig. 4 presents the time spent on the resources allocation services (new application and child tasks mapping and the secure zone creation, extension and releasing). First, it can be seen that the services execution time in a dynamic scenario is always greater. This is because in this case, there is more frequent activity from the controller by searching to extend or release the secure zones. Moreover, while the time to create a 5 cluster secure zone is greater than to create a 4 cluster one (execution scenarios a and b), the dynamic of the execution reverses the trend when there are several secure zones.

Resources utilization rate: Finally, the resources utilization rate

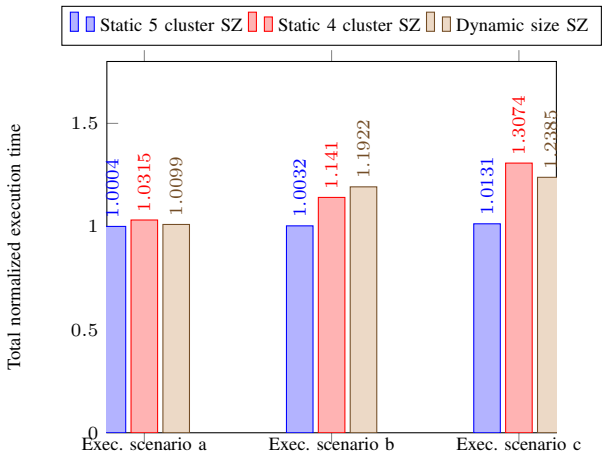

Fig. 3. Total execution time normalized to the baseline scenario

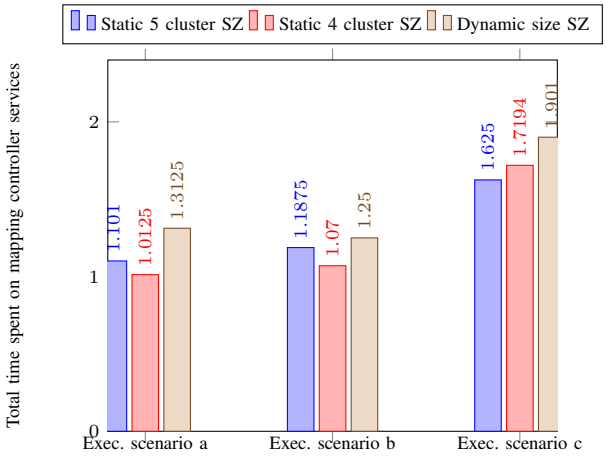

Fig. 4. Time spent on the controller services related to mapping (non-isolated new application mapping, new task mapping and SZ creation and releasing) normalized to mapping services time for the baseline scenario

in total as well as within secure zones are gathered in Table III. These depend on the activity of the platform and on the evaluated deployment strategy and execution scenario. Results show that as previously explained, when the secure zone is limited to 4 clusters, the resources utilization rate is lower than for a 5 cluster resources zone because of the higher isolated application execution time. Furthermore, dynamically resizing the secure zone achieves the highest resources utilization rate (72% for 3.a scenario, compared to 77% for the baseline scenario).

C. Discussion

Experimental results on the evaluated scenarios show that the performance overhead induced by the proposed secure-enable mechanisms increases with the number of required secure zones but still remains below 31% of the baseline execution time. It can be noticed as well that the performance overhead depends on the load of the platform when creating the secure zones and is negligible when there is no load on the platform (up to 3%, execution scenario b Fig. 3).

Moreover, the dynamic adaptation of the size of the secure zones offers the maximum resources utilization rate (up to 92% within secure zones and 72% in total Table III) and the minimum performance

Evaluated deployment strategy and execution scenario	Total exec. time	Exec. time in average for non-isolated applications	Exec. time in average for isolated applications	Exec. time in average
baseline	**363.35**	**202**	**-**	**202**
(1.a)	363.49	206	163	197
(1.b)	364.51	338	214	313
(1.c)	368.10	242	181	205
Average for 1.		262 (+29%)	186 (-8%)	237 (+17%)
(2.a)	374.79	261	198	248
(2.b)	414.58	264	269	265
(2.c)	374.51	272	270	271
Average for 2.		265 (+31%)	245 (+21%)	261 (+29%)
(3.a)	366.94	223	159	210
(3.b)	433.18	261	400	253
(3.c)	450.00	214	376	264
Average for 3.		232 (+14%)	311 (+53%)	242 (+19%)

TABLE II
COMPARISON OF THE TOTAL EXECUTION TIME AND EXECUTION TIME IN AVERAGE **(IN MSEC.)** FOR NON-ISOLATED, ISOLATED AND IN AVERAGE RESPECTIVELY FOR DIFFERENT DEPLOYMENT AND EXECUTION SCENARIOS

Evaluated deployment strategy and execution scenario	SZ resources utilization rate during the time of the SZ	Total resource utilization rate in average
baseline	**-**	**77%**
(1.a)	85%	68.5%
(1.b)	85%	71%
(1.c)	85%	61.6%
(2.a)	65%	64%
(2.b)	65%	69%
(2.c)	65%	55%
(3.a)	85%	72%
(3.b)	89%	69%
(3.c)	92%	67%

TABLE III
COMPARISON OF THE RESOURCES UTILIZATION RATE OF DEDICATED RESOURCES DURING THE SECURE ZONE (SZ) DURATION AND IN TOTAL FOR DIFFERENT DEPLOYMENT AND EXECUTION SCENARIOS

overhead for non-isolated applications (+14% in average Table II). However, this approach penalizes the performance of the isolated secure zones (up to 53% compared to the average for the baseline scenario), which explains the greater total execution time. While the first strategy (1) leverages the performance of isolated applications, it offers an average resources utilization rate. Strategy 2 penalizes both, the performance overhead on isolated and non-isolated applications in a balanced way but entails a worse resource utilization rate. Finally, the dynamic secure zone strategy, leverages the performance of non-isolated applications and offers the best resource utilization rate.

VII. CONCLUSION

In this paper an approach for the isolated execution of sensitive applications on a many-core accelerator is presented. A set of services able to dynamically create and release spatially isolated secure zones are proposed. This approach allows guaranteeing the isolated execution of sensitive applications avoiding the security vulnerabilities caused by sharing resources. Different deployment strategies for the creation of secure zones on several execution scenarios have been evaluated through virtual prototyping. According to the chosen deployment strategy, whether the isolated applications performance or the non-isolated applications performance can be penalized. Moreover, dynamically resizing the secure zones in order to adapt the dedicated resources allows achieving the highest resource utilization rate (72% compared to 77% for the baseline scenario) and the minimum performance overhead on non-isolated applications (down to +14%). Other secure zones deployment strategies as well as NoC protection will be considered in future work.

REFERENCES

[1] "TSAR," https://www-soc.lip6.fr/trac/tsar.
[2] "MPPA," http://www.kalrayinc.com/kalray/products/.
[3] Y. Wang and G. Suh, "Efficient timing channel protection for on-chip networks," in *Proc. of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip (NOCS)*, 2012, pp. 142–151.
[4] J. Demme and S. Sethumadhavan, "Side-channel vulnerability metrics: Svf vs. csv," in *Proc. of 11th Annual Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, 2014.
[5] J. Sepulveda, G. Gogniat, C. Pedraza, R. Pires, W. Chau, and M. Strum, "Noc-based protection for soc time-driven attacks," *Embedded Systems Letters, IEEE*, vol. 7, no. 1, pp. 7–10, March 2015.
[6] "ARM TrustZone," www.arm.com/products/processors/technologies/trustzone.
[7] G. Kornaros, I. Christoforakis, O. Tomoutzoglo, D. Bakoyiannis, K. Vazakopoulou, M. Grammatikaki, and P. Ao, "Hardware support for cost-effective system-level protection in multi-core socs," in *Proc. of Digital System Design (DSD)*, 2015.
[8] J. Porquet and C. Schwarz, "Noc-mpu: a secure architecture for flexible co-hosting on shared memory mpsocs," in *Design, Automation & Test in Europe (DATE)*, 2011.
[9] L. Fiorin, G. Palermo, and S. Co, "A security monitoring service for nocs," in *Proc. of 6th International conference on Hardware/Software codesign and system synthesis (CODES+ISSS)*, 2008, pp. 197–202.
[10] R. Masti, D. Rai, C. Marforio, and S. Capkun, "Isolated execution in many-core architectures," in *Proc. of Network and Distributed System Security Simposium (NDSS)*, 2014.
[11] "Open Virtual Platforms Imperas Software Limited, OVP & SystemC," http://www.ovpworld.org/technology_systemc.
[12] P. Wehner, J. Rettkowski, T. Kleinschmidt, and D. Göhringer, "Mpsocsim: An extended ovp simulator for modeling and evaluation of network-on-chip based heterogeneous mpsocs," in *Proc. of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2015.
[13] Omitted for blind review.
[14] "Xilinx Inc.,Zynq-7000 All Programmable SoC," www.xilinx.com/products/silicon-devices/soc/zynq-7000/.