Activation of Secure Zones in Many-core Systems with Dynamic Rerouting

Luciano L. Caimi*†, Vinicius Fochi†, Eduardo Wachter†, Daniel Munhoz†, Fernando G. Moraes†
*UFFS – Av. Fernando Machado 108E, 89802-112, Chapecó, Brazil
lcaimi@uffs.edu.br
†FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
{vinicius.fochi, eduardo.wachter, daniel.munhoz†}@acad.pucrs.br, fernando.moraes@pucrs.br

***Abstract*—Many-core architectures provide massive parallelism and high performance to the users. They also introduce key challenges regarding security. The main threat rises from the resource sharing. Adoption of firewalls, encryption mechanisms and resource isolation (processor or memory) are common strategies to treat the security threats. The two first strategies present high hardware cost, while the resource isolation does not protect the communication infrastructure. This paper proposes to protect communication and computation simultaneously, creating continuous *Secure Zones* (SZ) at runtime. The SZ is an isolated area in the system, preventing traffic flows to cross the boundaries of the zone, reserving the Processing Elements (PEs) inside the SZ to execute a secure application (App_{sec}). The App_{sec} traffic is enclosed inside the SZ, and any other traffic crossing the SZ is rerouted to the outside of the SZ. Experiments evaluate the cost to close and open an SZ, the impact in the packet latency with and without the SZ in the presence of malicious flows, and the performance penalty in non-secure applications due to the rerouting process when an SZ is closed. Results show that: (1) the latency to start an App_{sec} is small, 1.2 μs for an SZ with 25 PEs; (2) the isolation process prevents Deny-of-Service (DoS), timing, and spoofing attacks and guarantees confidentiality and integrity; (3) the overhead in the non-secure applications' execution time is also small, being the worst-case 2.56% longer.**

Keywords*—*Many-core, MPSoC, Network on Chip, Security.

I. Introduction

The many-core architecture assumed in this work are NoC-based MPSoCs, due to their inherent scalability. In such systems, several applications execute simultaneously, sharing computation and communication resources. A secure application, App_{sec}, that processes sensitive data may have its security harmed by a malicious process. Examples of attacks on such systems [1][2][3] include DoS, timing, spoofing, side-channel attacks.

A strategy to add security in NoC-based MPSoCs is the adoption of firewalls [1][2], to filter incoming and outcoming data according to some security policy. Another approach is to encrypt the data exchanged between processing elements (PEs), creating secure channels [1][4]. Temporal and spatial isolation of resources (processor and memory) are also used to protect applications from data information leakage [3][5]. Such methods protect the communication (firewall and cryptography) or the computation (temporal and spatial isolation).

The execution of an App_{sec} comprises at least three assumptions. The 1st one is the secure admission of the application, to guarantee the object code integrity. The 2nd assumption regards the application execution in an environment protected from attacks. The 3rd assumption is related to the protection of the communication with peripherals and shared memories.

The *goal* of this paper is to present a method to set Secure Zones (SZ) at runtime, enabling temporal and spatial isolation of applications. All traffic that should traverse the SZ is rerouted at runtime. Secure admission is partially addressed, and communication with peripherals and memories are out of the scope of the current work.

The *original contribution* of the proposal is the complete isolation of the application, protecting computation and communication resources from attacks. The proposal isolates a given system region by using wrappers, instead of firewalls, reducing the hardware cost. Also, as the application is isolated, there is no need to encrypt packets, reducing the communication latency between application's tasks.

II. State-Of-the-Art

Sepúlveda et al. [1] and Isakovic et al. [4] protect communication and computation, adopting firewalls and cryptography. Sepúlveda et al. [1] adopt two NoCs: a service (used to security control packets) and a data NoC. This proposal set SZs at runtime, encrypting the packets. Packets from other applications may traverse the SZ, making the approach vulnerable to timing attacks. Isakovic et al. [4] propose an architecture partitioning of the MPSoC at design time. The Authors adopt security components like a *secure μKernel* and a *secure channel* infrastructure.

Hu et al. [2] and Fernandes et al. [6] protect, at design time, only communication while the computation is exposed to resource sharing. Hu et al. [2] select the position of the firewalls to reduce the overhead for security information in packets headers. Fernandes et al. [6] propose the creation of SZs based on the routing algorithm. The Authors use the Region-based Routing to create SZs and the Segment-based Routing to guarantee deadlock free paths. The presented results mitigate DoS and timing attacks.

Real et al. [3] and ARM TrustZone [5] protect only computation. Both approaches are applied at runtime. Real et al. [3] use a clustered MPSoC interconnected by routers. The Authors create SZs, mapping the application to one or more clusters. If an application task needs to communicate with a task in another cluster, the message is sent through an insecure channel. ARM provides the TrustZone [5], hardware support for the creation of secure environments, isolating applications in the same processor. Applications running on different processors share the communication infrastructure and memory. Thus, these processors are not protected from each other since sharing the communication infrastructure leads to possible leakage of information attacks.

Our proposal protects communication and computation simultaneously, creating continuous SZ at run-time, by reserving PEs and communication channels to execute the App_{sec}. The App_{sec} traffic is enclosed inside the SZ, and all other traffic is deviated from the SZ.

III. System Architecture

Figure 1 presents the baseline architecture, a homogeneous NoC-based MPSoC, where each PE contains a 32-bit RISC processor, a DMNI module (a network interface with DMA capabilities) and a local dual-port memory accessed by the processor and DMNI module. The software executing at each PE defines its role in the system. The system has manager PEs, global manager (GMP) and local managers (LMP); and PEs executing applications, slave processors (SP).

Two NoCs interconnects the PEs: *data* and *control* NoC. The data NoC transfers *data messages*, exchanged by applications. The data NoC adopts duplicated physical channels, wormhole packet switching, simultaneous support for distributed XY and source routing. Duplicated physical channels ensure full routing adaptivity.

The control NoC transfers the *control messages*, such as: *(i)* fault notification; *(ii)* cache invalidation; *(iii)* the set of PEs belonging to a SZ; *(iv)* definition of a fault-free path when a given part of the data NoC fails. The control NoC has the following features (similar architecture to [7]): adoption of broadcast as the default transmission mode, bufferless router, each message has one flit. The control NoC router has a small area footprint, corresponding roughly to 20% of the data router.

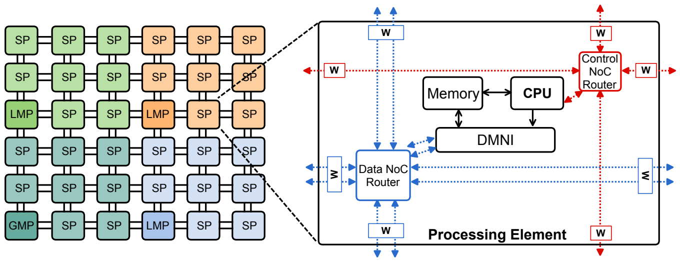

Fig. 1. System Architecture. Wrappers (W) are added to the control signals of NoCs links, enabling to isolate ports individually.

Both NoCs contain test wrappers, or simply *wrappers*, in the flow control signals. When activated, the wrapper enables to discard all incoming and outcoming packets of a given port. This approach guarantees the creation of a physical barrier at the hardware level, and thus the SZ creation. A wrapper contains a flip-flop and two 2x1 multiplexers. Firewalls [8][9] are much more complex than wrappers since they require memory to store the secure rules and state machines to control the incoming and outcoming traffic. In the current work, the Operating System (OS) of each PE controls the wrappers connected to the data NoC. The control NoC manages their wrappers, for security reasons, i.e. the applications running at the PEs cannot access the wrappers of the control NoC.

The current work uses the control NoC to transfer the control messages to close and open a SZ. The control NoC has two operation modes: *global* and *restrict*. The *global* mode enables the control messages to pass through the wrappers, even if they are enabled. This mode enables the PEs inside the SZ to exchange messages with manager PEs. The *restrict* mode observes the status of the wrappers. If a control message hits an activated wrapper, it is discarded, corresponding to a barrier to the broadcast transmission.

The data NoC observes the status of the wrappers. A data message arriving in an activated wrapper is always discarded, and the control NoC returns to the source of the message a new broadcast reporting that the message needs retransmission. This enables the PEs outside the SZ to search a new path.

IV. Threat Model

The resource sharing of MPSoCs components introduces vulnerabilities to App_{sec}s running on it. It is possible to explore these vulnerabilities in attacks, such as:

- *Confidentiality and integrity*: unauthorized access to the data by writing or reading. With different applications sharing the MPSoC, a malicious application can be loaded and executed by a given processor, accessing the memory to retrieve or leak critical data;
- *Denial of Service - DoS*: the goal is to disrupt the system by overloading resources. A malicious application generating packets with a high injection rate can produce this attack, overloading the communication infrastructure.
- *Timing attack*: explores the communication collision between the sensitive traffic and the attacker traffic. The latency interference induced by malicious traffic can provide to the attacker some information about the timing, frequency, and volume of the secure communication.
- *Spoofing*: a malicious application successfully falsifies its identity to obtain unauthorized privileges.

Our proposal adopts two *fundamental assumptions*: *(i)* a SZ is a continuous region, with a square or a rectangular shape; *(ii)* there is no resource sharing inside the SZ, i.e., only one application executes inside the region. The activation of the wrappers at the boundary of the SZ enables the first assumption. The mapping procedure enables the second assumption. Once defined the SZ, tasks executing in the SZ are migrated to outside the SZ before the App_{sec} mapping.

The adoption of these assumptions avoids the attacks previously presented. As only the App_{sec} executes in the SZ, and the wrappers block the traffic at the boundary of the region, any access or data modification by an external application is blocked, ensuring data confidentiality and integrity. Timing or DoS attacks to the App_{sec} are automatically refused by the wrappers since no external traffic is allowed inside the SZ. Thus, malicious applications cannot extract any information with temporal correlation to a task running inside the SZ, or overload the resources (routers and processors). In the same way, spoofing attacks are prevented since data cannot cross the SZ boundary. Side channel attacks, as power-monitoring or electromagnetic attacks, are not considered in this proposal. However, the feasibility of such attacks in a system with dozens of processors is unlikely to occur.

V. Proposed Method to Set Secure Zones

The proposed method includes: *(i)* admission of the App_{sec}; *(ii)* reroute of packets outside the SZ; *(iii)* retransmission of lost packets in and out the SZ boundaries; *(iv)* launch App_{sec}. Figure 2 illustrates the method. In Figure 2(a) the MPSoC contains one application in execution, app_1. Next, the LMP maps an App_{sec}, activating the wrappers at the boundary of the SZ. At this moment (Figure2(b)), the app_1 traffic is blocked by the SZ. Figure 2(c) shows the App_{sec} executing in the SZ, and the traffic of app_1 circumventing the region.

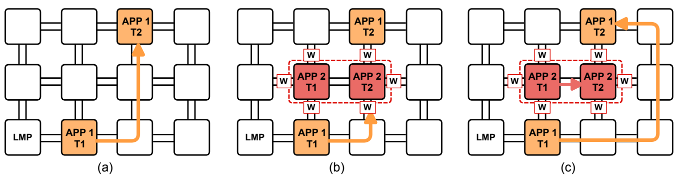

Fig. 2. Secure zone and dynamic reconfiguration of routing paths.

A. Protocol to Close a Secure Zone

Our approach assumes that applications can be loaded to the MPSoC at runtime (dynamic workload). The external environment of the MPSoC requests to the GMP the admission of new applications. An App_{sec} has a Message Authentication Code (MAC) appended at the end of the object code of each task to avoid malicious corruption while loaded into the MPSoC, enabling the secure admission of the application.

Figure 3 details the main steps of the protocol. The labels in the right side of the Figure corresponds to: (**a**) admission of App_{sec}; (**b**) close SZ; (**c**) clear memories and open SZ.

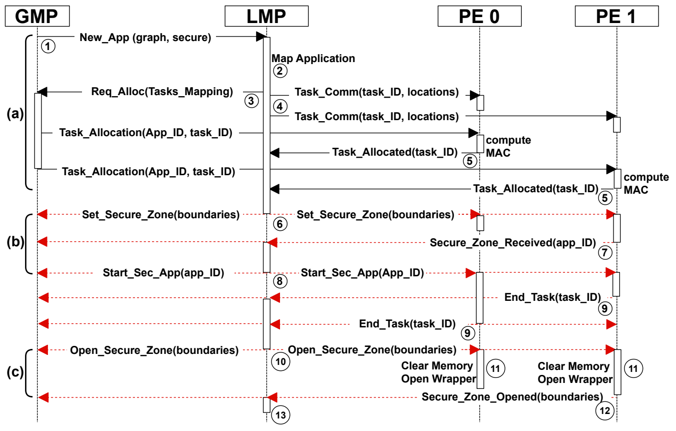
GMP
LMP
PE 0
PE 1
New_App (graph, secure)
Map Application
Req_Alloc(Tasks_Mapping)
Task_Comm(task_ID, locations)
Task_Comm(task_ID, locations)
Task_Allocation(App_ID, task_ID)
compute MAC
Task_Allocated(task_ID)
Task_Allocation(App_ID, task_ID)
compute MAC
Task_Allocated(task_ID)
Set_Secure_Zone(boundaries)
Set_Secure_Zone(boundaries)
Secure_Zone_Received(app_ID)
Start_Sec_App(app_ID)
Start_Sec_App(App_ID)
End_Task(task_ID)
End_Task(task_ID)
Open_Secure_Zone(boundaries)
Open_Secure_Zone(boundaries)
Clear Memory
Open Wrapper
Clear Memory
Open Wrapper
Secure_Zone_Opened(boundaries)
(a)
(b)
(c)

Fig. 3. Protocol to set a SZ. Messages with a dashed line are sent through the control NoC

When the GMP receives the request to execute an App_{sec}, it selects a cluster to execute the App_{sec}, transmitting the application graph to the LMP of the selected cluster (event **1** in Figure 3). The LMP executes the task mapping heuristic (**2**) considering the assumptions related to App_{sec}: a continuous region with a rectangular shape. The task mapping heuristic defines the *location* of each App_{sec} task. If there are tasks executing inside the region defined by the mapping heuristic, the LMP migrates them to outside the SZ to guarantee *confidentiality* and *integrity*. Once defined the tasks' locations, the LMP requests to the GMP to transmit the object code of the tasks to the SPs (`Req_Alloc` message, **3**). Also, the LMP transmits to the SPs the addresses of the communicating tasks (`Task_Comm` message, **4**). Each PE receiving a task object code computes the MAC, comparing it to MAC appended at the end of object code. Each PE returns a `Task_Allocated` message (**5**), with the result of the MAC comparison. If no error is detected in the MAC comparison, the protocol continues. Otherwise, the GMP aborts the admission of App_{sec}. It is worth to mention that the received tasks are *not* scheduled to execute. The tasks are received by the SPs and remain blocked by the OS, up to a release message sent by the LMP.

The admission of non-secure application differs from the admission of an App_{sec} in three points: *(i)* the mapping region may be discontinuous; *(ii)* there is no need to migrate tasks sharing the PEs with the tasks of the application being mapped; *(iii)* there is no MAC to check.

When all PEs returns a `Task_Allocated` message, the LMP can close the SZ. The LMP broadcasts a `Set_Secure_Zone` message (**6**) through the control NoC with the upper right and lower left corners of the SZ in the payload. All PEs receive this message, each one verifying if it is on the SZ boundary. If the PE is on the SZ boundary, the OS writes in a data structure the wrappers to be closed. The PE located at the upper right corner of the SZ transmits to the LMP a `Secure_Zone_Received` message (**7**). Once received this message, the LMP can start the execution of App_{sec}, by broadcasting a `Start_Sec_App` message (**8**), using the control NoC in *global* mode. This message enables the PEs to activate the wrappers to block incoming/outcoming traffic and releases the tasks belonging to App_{sec}.

At this point, the wrappers discard all flows traversing the SZ. The control NoC transmits to the source of the discarded messages (*restrict* mode) a message retransmission request. The non-secure applications use the control NoC to find an alternative path to circumvent the SZ and retransmit the non-delivered message. The default routing algorithm is the XY. Alternative paths use source routing. Deadlock prevention and full-adaptivity are ensured by using two disjoint networks (2 physical channels per link). Note that the alternative path computation is executed once, only for the first lost message. Thus, the impact of closing an SZ in the non-secure applications is minimal, as shown in the results section.

B. Protocol to Open a Secure Zone

When a App_{sec} task ends its execution (**9**), the PE sends a `End_Task` message with its task ID, using the control NoC in *global* mode. When all App_{sec} tasks finish their execution, the LMP transmits an `Open_Secure_Zone` message (**10**). All PEs inside the SZ clear their memory to prevent information leakage and then open the wrappers (**11**), releasing the reserved resources. Then, the PE located at the upper right corner of the SZ transmits to the LMP a `Secure_Zone_Opened` message (**12**). Finally, the LMP clear the internal structures to release the cluster resources previously allocated to App_{sec} (**13**).

VI. Results

The **first** evaluation concerns the impact to close and open an SZ in 5 scenarios, changing the SZ size from 2x1 up to 5x5 PEs. The evaluation considers the steps presented in Figure 3 (labels at the right side of the Figure). Table I presents the evaluation, considering the SZ size. Results for *Task Allocation* (2nd row of the Table) are a function of the App_{sec} size, and therefore constant for this experiment. The number of clock cycles to create a SZ ((b) in Figure 3) starts when the LMP sends the `Set_Secure_Zone` message, until the upper right PE in SZ effectively activate theirs wrappers after the `Start_Sec_App` message. The *Close Secure Zone* results (3rd row) present a small increase in the number of the clock cycles, proportional to the distance to the upper right PE of the SZ. The time to close a SZ ((c) in Figure 3) starts when the LMP sends the `Open_Secure_Zone` message until the reception of a `Secure_Zone_Opened`. The *Open Secure Zone* delay (4th row) is related to the amount memory pages to erase. All memory pages (64 KB in the experiment) used by App_{sec} are erased at this protocol step.

TABLE I. CLOCK CYCLES TO EXECUTE THE PROTOCOL TO SET A SZ.

Interval / Size of SZ	2x1	2x2	3x3	4x4	5x5
(a) Task Allocation	10298	10308	10466	10597	10704
(b) Close Secure Zone	988	1028	1097	1166	1186
(c) Open Secure Zone	5943	5962	6006	6048	6081

The result revealed by the Table is the *scalability*. The *Task Allocation* delay is the same for secure and non-secure applications. The overhead induced by the method comes from the *Close Secure Zone*, which corresponds to less than 1,200 clock cycles to a large SZ (1.2 μs@100MHz). The *Open Secure Zone* does not impact in the applications' execution time. The *Open Secure Zone* delay affects the amount of time to release the PEs, which is also small (few μs).

The **second** evaluation presents the intrusiveness of non-secure applications in an App_{sec}. For this experiment, we propose a scenario where an App_{sec} shares communication resources with malicious flows. Figure 4(a) presents the mapping configuration: an App_{sec} mapped in the SZ (surrounded by the doted line) and six other tasks mapped outside the SZ generate malicious flows. The latency of the packets on the north port of the PE executing *TaskD* is measured according to 3 scenarios (Figure 4(b)): (a) App_{sec} executing alone in the system; (b) all tasks running without the SZ; (c) all tasks running with the SZ activated. Scenario (a) produces the baseline latency between taskD and taskF, without any disturbing traffic. In the second scenario (b), the malicious flows disturb the taskD latency. The 3rd scenario shows that the application running inside the SZ the latency has the behavior of the baseline scenario, *preventing* DoS or timing attacks.

The **third** evaluation presents the impact of the SZ on non-secure applications. Close a region inside the MPSoC at runtime implies that applications executing outside of SZ must continue the execution, by reconfiguring and rerouting the source-target paths. Figure 5 presents the mapping of two non-secure applications. The dotted squares in the Figure represent the boundary where the App_{sec} can be mapped and executed.

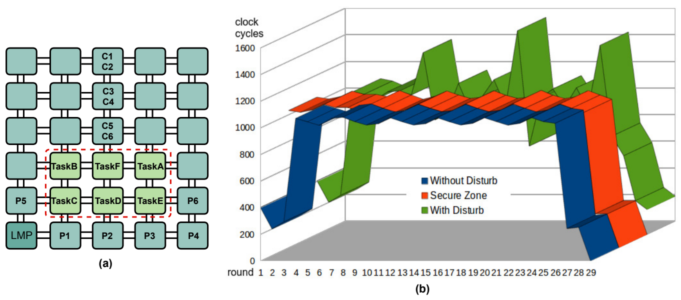

Fig. 4. Communication latency: (a) App_{sec} in the SZ, and malicious tasks outside the SZ; (b) latency graph results.

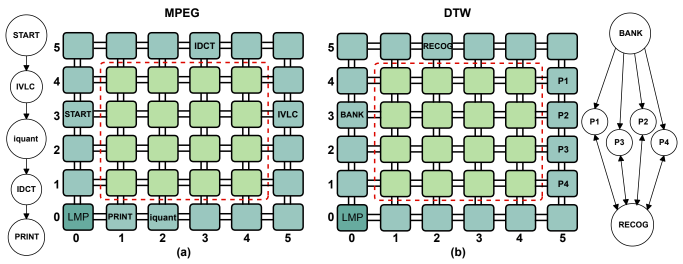

Fig. 5. Task graphs and mapping. (a) MPEG and (b) DTW applications.

The non-secure applications start their execution, and at a given moment the App_{sec} is mapped, and then the SZ closed. For each non-secure application, 6 secure zones scenarios are evaluated. Table II presents each scenario, the non-secure application execution time (AET) overhead (2nd and 4th columns), and the number of rerouted paths (3rd and 5th columns), for MPEG and DTW applications, respectively.

TABLE II. IMPACT OF THE SZ IN THE NON-SECURE APPLICATIONS.

Secure Zone Boundary	MPEG AET overhead	MPEG # Paths Rerouted	DTW AET overhead	DTW # Paths Rerouted
((1,2),(3,4))	1.22%	7	2.53%	11
((2,2),(3,4))	1.13%	6	2.53%	11
((1,1),(3,4))	1.22%	7	2.56%	12
((3,2),(3,4))	1.13%	5	2.53%	10
((1,3),(4,3))	1.17%	7	0.98%	8
((1,2),(4,3))	1.17%	7	0.79%	9

The execution time overhead of the applications outside of the SZ is in the worst-case 2.56%. This overhead is related to the number of rerouted paths, which is a function of the SZ location, and the number of interrupted flows. For example, the third DTW scenario (highlighted in gray) has a large SZ (12 PEs), requiring the execution of 12 rerouting paths. *This experiment showed that SZ might be defined at runtime, with a negligible impact on the performance of the applications outside of the region.*

VII. CONCLUSION

This work presented a method to set Secure Zones at runtime in NoC-based many-core systems using wrappers, a low-cost hardware mechanism. The secure zone completely isolate the application, protecting computation and communication resources from attacks (integrity, confidentially, DoS, timing, and spoofing) from malicious applications.

Results show the effectiveness of the approach and the negligible impact in the execution time of secure and non-secure applications. Future works comprise to enhance the secure admission protocol of secure applications and the protection of the communication with peripherals and memories.

ACKNOWLEDGEMENT

The Author Fernando Moraes is supported by CNPq and FAPERGS funding agencies.

REFERENCES

[1] J. Sepúlveda *et al.*, "Reconfigurable security architecture for disrupted protection zones in NoC-based MPSoCs," in *ReCoSoC*, 2015, pp. 1–8.

[2] Y. Hu *et al.*, "Automatic ILP-based Firewall Insertion for Secure Application-Specific Networks-on-Chip," in *INA-OCMC*, 2015, p. 4.

[3] M. M. Real *et al.*, "Dynamic spatially isolated secure zones for NoC-based many-core accelerators," in *ReCoSoC*, 2016, pp. 1–6.

[4] H. Isakovic and A. Wasicek, "Secure channels in an integrated MPSoC architecture," in *IECON*, 2013, pp. 4488–4493.

[5] ARM, "ARM Security Technology Building a Secure System using TrustZone Technology," 2008. [Online]. Available: http://infocenter.arm.com

[6] R. Fernandes *et al.*, "A security aware routing approach for NoC-based MPSoCs," in *SBCCI*, 2016, pp. 1–6.

[7] E. Wachter *et al.*, "Topology-Agnostic fault-tolerant NoC routing method," in *DATE*, 2013, pp. 1595–1600.

[8] R. Fernandes *et al.*, "A non-intrusive and reconfigurable access control to secure NoCs," in *ICECS*, 2015, pp. 316–319.

[9] H. K. Kapoor *et al.*, "A Security Framework for NoC Using Authenticated Encryption and Session Keys," *Circuits, Systems, and Signal Processing*, vol. 32, no. 6, pp. 2605–2622, 2013.