The CSI Journal on
Computer Science and Engineering
Vol. 16, No. 2, 2019
Pages 54-68
Regular Paper

A Non-interference Isolation Mapping to Eliminate Timing Channel Attacks

Mohammad Sadegh Sadeghi Siavash Bayat Sarmadi Shaahin Hessabi

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Abstract

On-chip network security is a dominant necessity for secure many-core platforms to leverage the concepts of the cloud and embedded system-on-chip. In fact, there is a need for a secure, low-latency and highly utilized on-chip communication. While there are various techniques to mitigate and eliminate timing side-channel attacks, these attacks present a security threat for networks-on-chip (NoCs). This threat stems from the experienced latencies of the malicious application, which can divulge information when the victim applications are accessing shared resources. We propose an approach to eliminate the timing side-channel attacks, using isolation dynamic mapping. In order to meet the non-interference flows between malicious and victim applications and the security of the NoC, our proposed schemes (Isolated and Liso) employ several techniques, including (1) a scalable strategy that maximizes utilization of the system, and (2) a low-overhead isolation approach for the NoC through rectangular and L-shape mappings. We show that Isolated mapping scheme degrades the throughput of our secure NoC system while Liso scheme improves it without leaking information. Our best solution offers throughput that is on average 16\% lower than that of an optimized non-secure baseline, and has no router modification in contrast to the best known competing schemes.
Keywords: : Non-interference, Security, On-Chip Networks, Timing Side-Channel Attacks.

1. Introduction

Today, network-on-chip (NoC) is considered as an appropriate interconnection in many-core platforms. It provides increasing performance and throughput by means of shared resources. However, resource sharing may present some security risks. One of the most important security threats is known as timing channel attack. It has shown that timing channel exploits arise from any shared hardware components: shared cache [1]–[4], shared memory [5]–[8], and on-chip networks [9], [10].

Ristenpart et al. [11] show such possible cache-based side-channel attack on Amazon EC2 hardware to recover user passwords. The formation of timing side-channel attack can occur through two scenarios. Firstly, attacker may simply discover secrets without knowing the identity of the co-resident victim. Secondly, the attacker may deliberately attempt to obtain co-residency with a specific target [11].

This security threat can cause leakage information to malicious applications, intentionally or unintentionally. The former kind of leakage is known as covert channel, while the later is known as side channel. When two applications reside together in the network in such a way that the flow of one's information packets interferes with the other one, there is a possibility of inferring some information by malicious application through estimating of its throughput.

The problem is that NoC has many internal resources that are shared between packets from different domains. These resources include the buffers holding the packets, the crossbar switches, and individual ports and channels. Resource contention among these shared resources may incur interference. Interference has been known as the main cause of timing side-channel attacks. In this work, our focus is on isolation of applications in order to completely eliminate the timing side channels in the NoC rather than showing how to actually construct them. We do not assess the possibility of timing side-channel attack itself since it has been adequately covered by prior work [9], [10]. NoC-based timing side channel can be inaccurate due to the noise, which can be caused by frequent use of the shared resources by many

applications other than the malicious and victim applications [11]. Regardless of being noisy, such information can be useful for clever attackers. There are some possible ways to obtain weak signals from strong noises, aggregated information from low-rate leakage and correlate leakage across multiple channels. Also, it has been shown that there are some amplification techniques, which are functional for arbitrary timing channels [12]. Our proposed method does not cover cache-based and memory-based side-channel attacks, and focuses on NoC side-channel attacks.

Hard isolation techniques, as general solutions to prevent side-channel attacks, are used in modern virtualization techniques such as Xen [13], HyperV [14] and VMWare [15]. These techniques are the cornerstone for the security of computing systems [16]. However, hard isolation diminishes efficiency and raises cost because of poor resource management [17]. In contrast, we use soft isolation in order to maintain, as much as possible, the nature of resource sharing of the NoC.

There are three alternative schemes, which can be used to eliminate network interference in the NoC: static partitioning, temporal partitioning and composability. Static partitioning allocates resources as pre-specified budget. Instead of allocating routers to security levels, temporal partitioning provides non-interference through allocating virtual channels (VCs) in each input port to security domains. Another way to completely eliminate interference between applications in the system is to use composability [18]. However, static and temporal schemes incur significant performance overhead due to static allocation of resources without considering resource demands at run-time. Also, the third scheme has a significant area and power overhead due to using different components.

NoC virtualization is another strategy for applying isolation in order to eliminate interferences. This strategy is based on dynamic resource partitioning. In order to implement this strategy, it uses reconfiguration mechanism to adapt the network resources in order to provide the application requirements. NoC-based virtualization is presented to improve the performance of the applications [19]. It does not propose any idea regarding the process of partitioning, and relies on a resource manager operating under OS control. It is based on the strategy proposed in [20] for assigning the resources to the applications, and moreover, it uses a first-fit approach for mapping process. Also, it uses segment-based routing (SR) algorithm [21] to support irregular shapes. These features increase complexity, and cause more performance overhead.

In this work, we push the envelope even further by introducing L-shape and Isolated mapping process to improve the security of NoC systems, and gain better performance. We concentrate on regular shapes in order to significantly mitigate the complexity by means of XY routing algorithm, and gain higher system throughput through supporting rectangle shapes. Our work shows that it is possible to completely eliminate timing side-channel attacks, and the proposed schemes provide non-interference among different information flows. Also, the simulation results indicate that the performance overhead of the proposed protection can be minimal.

1.1. Contribution

This paper makes the following contributions:

- Proposes dynamic isolation mapping schemes to afford sufficient adaptivity for efficient resource management and support both low-overhead performance and non-interference between security levels.
- Demonstrates the effectiveness of dynamic mapping process in order to establish security for the NoC, and thus provides non-interfering environment.
- Shows the advantage of proposed schemes by increasing network sizes and the number of applications especially for Isolated approach.

Additionally, we have evaluated the overhead of the system generated by Liso and Isolated approaches and show that they have 16% and 25% throughput overhead compared to an insecure baseline approach, respectively. Moreover, our approaches have no additional hardware overhead compared to the best known competing schemes.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. Section 3 describes the problem statement and security model. Our schemes are explained in Section 4. Section 5 presents simulation results of our schemes. Finally, Section 6 concludes the paper.

1. Related Work

To the best of our knowledge, this is the first proposed dynamic isolation mapping for a homogeneous on-chip network to eliminate timing side-channel attacks. Our proposal is related to other proposals of non-interference and timing channel protection techniques, and dynamic isolation for virtualization in networks-on-chip, which we review below.

1.1. Interference and Timing Channel in Networks-on-Chip

The outgrows of NoC, which offers higher density and performance, was emphasized by the reaching of Tilarea's Tile Gx100 [22] and Intel's 48-core SCC [7]. This has further necessitated the elimination of interference and timing channel in embedded and cloud systems. These many-core processors depend upon shared resources that immensely impacts application security, as described by Wang et al. [9]. This can lead to vulnerabilities in existing QoS techniques for NoC-based systems, such as the ones proposed by Grot et al. [23], [24], [25], and Lee et al. [26]. Wang et al. [9] demonstrated that even with giving quality-of-service guarantee, the vulnerability in NoC when sharing architectural resources is principal. Wassel et al. [10] proposed SurfNoC, an on-chip network that reduces the latency occurred by temporal partitioning. SurfNoC modifies network router design in order to provide packet switching and relies on static virtual channel scheduling.

Wassel et al. [10] also examined the impact of time-division multiplexing of resource sharing in NoC at the architecture level, reducing the effectiveness of techniques such as TDMA

[27] and proportional resource sharing [28] for real-time systems. Chen et al. [29] introduced a region-aware interference reduction technique to improve the average packet latency. For resource-sharing NoC-based systems, it is important to accurately eliminate the interference of applications and exercise control over it in order to maintain performance guarantees, resource utilization and improve security, as also pointed out by zhang [30] and Papastefanakis [31]. Our schemes provide a dynamic mapping for non-interference between applications from different domains. Such dynamic mapping can be useful for high-order decisions such as resource management and run-time optimization, as suggested by Fiorin et. al [32] and Motakis et al. [33].

1.2. Dynamic Isolation and Virtualization in Networks-on-Chip

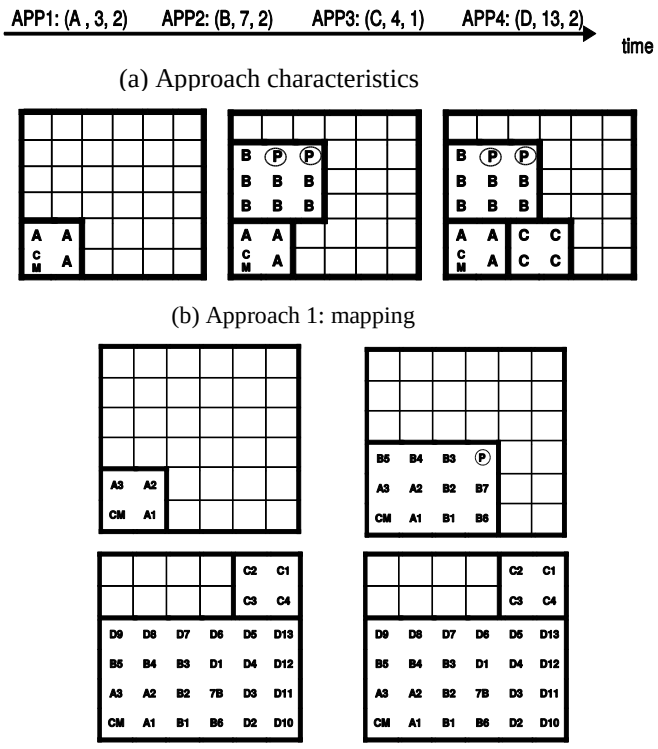
Dynamic isolation and virtualization techniques have been proposed for several purposes, but their motivations have been entirely different from ours. Dynamic isolation has been proposed as a means to reduce resource interference. For the first time, Flich et al. [34] proposed mechanisms for performance isolation for NoC to clarify the tradeoffs between topology regularity and routing complexity. The conception of our schemes is similar to the virtualization process proposed by Trivino et al. [19]; however, their technique provides irregular-shaped isolation based on complex routing algorithm and no information is given about resource management requirements such as shape and size of partitions, which are critical in performance considerations.

Lu et al. [35] also proposed a relaxed isolation scheme, which is used for improving performance isolation, but cannot eliminate NoC timing channels. Our first isolation scheme (Isolated) provides a strict NoC isolation strategy using rectangular shapes to avoid information leakage by innocuous applications and maintain performance. In order to enforce performance isolation constraint in workload consolidation, we also propose L-shape isolation (Liso) scheme to support a more flexible topology. This way, our scheme is able to achieve high consolidation density, while it does not complicate the routing mechanism. In other words, it uses efficient and cost-effective Dimension Order Routing (DOR) routing mechanism while it completely eliminates side channels and maintains security of domains.

Our proposed dynamic mapping mechanism can be used in conjunction with mechanisms that eliminate other side channels, such as memory and cache. Our approach is optimized for embedded and cloud systems, with emphasis on performance and side-channel attacks.

2. Assumptions, Security Model and Problem Statement

In this section, we present our main idea through an example and some definitions, discuss security model, describe timing channel protection, and state the problem that we address in this paper.



(a) Approach characteristics

(b) Approach 1: mapping

(c) Approach 2: L-shape mapping

Figure 1: Two mapping approaches.

2.1. Motivational Example

We illustrate our proposed mapping process to eliminate a timing side channel protection using four applications. It should be noted that our main goal in this paper is security achieved by non-interference; so we propose different approaches in this area. Before showing an example of our approaches, some terminologies need to be introduced.

Let APP (N, S, SL) characterize an application where N, S and SL are application name, size and security level, respectively. Also, "P" and "CM" symbols in Figure 1 are defined as "penalty node" and "control manager", respectively. Penalty node is an unused core in a security region during mapping process. Control manager is a master node. As shown in the isolated mapping in Figure 1(b), whenever an application enters the system, its objective is to find the smallest square on the platform such that all communication packets of the application are confined to their isolated region and minimum fragmentation occurs after the mapping. In this way, as defined in [36], minimum external network contention occurs. External contention happens when two flows from different applications contend for the same links. Lowering the external contention provides the benefit of decreasing the additional communication costs and execution time of the applications. However, the drawback of this approach is that with this contiguity constraint, the achievable throughput is degraded as a result of the increased turnaround time [37]. More details can be found in the example discussed in Figure 1. Figure 1(a) shows the characteristics of applications (Apps 1 through 4). More precisely, each application contains a number of tasks that will acquire their resources. Figures 1(b) and 1(c) show the result of application mapping after a specific time through

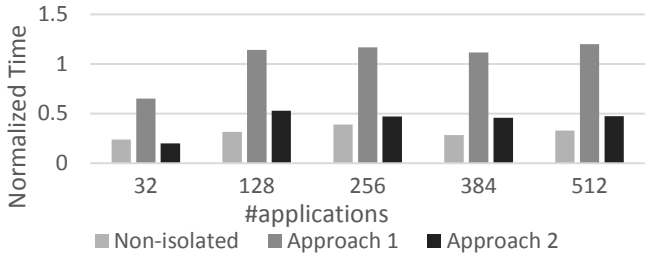


Figure 2: Normalized average turnaround time of applications for different workload sizes.

different approaches. These two different approaches are as follows.

- Approach 1: Through rectangle mapping, this approach attempts to isolate each application in order to minimize external contention and communication cost. In other words, it provides security per application individually.
- Approach 2: Through L-shape mapping, this approach attempts to make an isolated and rectangle region for a group of applications, which have the same security level in order to maximize throughput. In other words, it provides security per domain (a set of applications that do not need to be isolated from each other).

In this example, we assume that the XY-routing algorithm is used for communication. For Approach 1, the external contention is minimized for the mapping of three applications. However, in the best case, the remaining (available) resources could not fit for mapping the fourth application with size of 13, even though enough resources exist. In other words, throughput and utilization of the system decrease.

Our motivation for applying Approach 2 is to gain more throughput. This approach fits more applications on the system and subsequently decreases the turnaround time (total time between submission of an application and its completion) of each application. We have analyzed the turnaround time of a set of workloads for a network size of 100 nodes. The result is shown as a histogram in Figure 2. The result shows that the non-isolated approach (SHiC) [38] has the lowest turnaround time; therefore, we use SHiC as a baseline. Approach 2 has an improvement of about 60% compared to Approach 1. The basic idea is that, by considering different security levels, we can accumulate applications with the same security level, and therefore improve the system utilization and throughput. Although this may cause a larger external contention, it can be later mitigated by making more constraints for mapping applications in the same security level. We also observe that external contention increases system fragmentation, which has a huge impact on system throughput. This can be mitigated in our approach by filling fragmented resources for each recently entered application for a specific security level.

2.2. Security Model

In this model, we assume that a set of applications by the same security level can be placed together, but applications with different security levels should be placed separately. Accordingly, we define a set of applications with the same security level as a domain. This model is defined based on complete isolation of different security levels. Each new application is located in a domain appertain to its security level. All applications within a security level use shared resources, and are isolated from other applications with different security levels. In fact, mechanism of isolation comes from the nature of XY routing algorithm and applying rectangular mapping. Moreover, a logic-based distributed routing (LBDR) [34] implementation makes a boundary around each domain dynamically. According to this methodology, we can ensure that all three requirements of security (confidentiality, integrity and availability) have been maintained. Although our security model is extensible for any number of security levels, for simplicity, we present it for one to four and *n* security levels.

In this work, we assume that attacker can share its information flows with arbitrary flows of other security levels through shared resources. Therefore he/she can measure the latency of his/her packet transmissions. Also, we focus on non-interference policy of information flows. This policy is defined as security levels or domains through the NoC. According to this security model, we present a system, which assures that information flow conform to the specified policy; that is, no private information (secret) may leak to a malicious application. This security model includes timing channel protection and does not include attacks such as physical (EM emission, temperature, etc.).

Our security model protects against the following threats:

1. Confidentiality- malicious application plans for discovering information through information flows, which are shared by NoC.
2. Integrity- execution time of victim's application can be manipulated through information flows of malicious application. In fact, our schemes close illegal information flows through the NoC and enforce strict non-interference.
3. Availability- interfering information flows of victim's application with malicious application are susceptible to DoS attacks.

We trust produced devices of the NoC. We assume that all requests and replies through the NoC have been routed correctly. Also we trust peripheral devices and memories.

2.3. Timing Side-Channel Protection

There is a possibility of interfering flows in the NoC in such a way that a malicious application can evaluate the timing of information flows for discovering some secrets. This way, a malicious application is able to measure the time variation and throughput of its information flows in order to infer some information. This leakage information is termed as a side-channel attack. As an example, if a malicious application plans for this attack, it can transmit its packets consecutively, thus the possibility of interference can occur on a crossbar switch. Accordingly, the run time of a malicious application is affected by the dependency of conditional decisions, which may happen in a target application, for example RSA algorithm. Therefore, attacker can estimate the number of 0 or 1 transmissions from the target application. Moreover, a side-channel attack can happen based on traffic analysis. In fact, the timing pattern of traffic can be learned by the attacker through measuring the queuing in crossbar and buffer

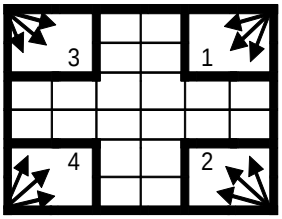


Figure 3: Pattern of mapping process for L-shape mapping.

input/output ports.

Another important security issue can be caused by interference among cores during dynamic mapping, which we refer to it as timing interference via core mapping. This may happen through private resources of each core. The resources that are dedicated to each core are only used by one domain at a time. However, multiple domains can use these resources through dynamic allocation. Therefore, timing channels exist if the state is kept across context switches. Hence, in order to eliminate this timing channel, domains flush the per-core state when a core leaves a domain. To prevent information leakage, the time that these flushing operations take cannot depend on the domain's state. For example, cache flushing should not take longer when there are more dirty blocks. Therefore, after flushing, each core is blocked until the worst case writeback time is passed.

To prevent writeback requests from interfering with the incoming process, the core must be stalled until all writebacks are complete. The time required to block the pipeline depends on the worst-case time that is required to drain all writebacks. The worst-case writeback time can be determined by assuming that every cache block in each part of the cache hierarchy allocated to the sum of the outgoing L-shape region in Liso approach or rectangle shape region in Isolated approach is dirty, and needs to be written back. Meanwhile, total cores of an L-shape region at the border of a domain or rectangle shape are not necessarily occupied. In other words, in most of the cases, some of these cores are not occupied. A rough approximation can be determined by multiplying the size of each private cache or shared cache partition allocated to this L-shape region at the border of each domain or rectangle shape by the latency of the cache one step up in the hierarchy. For the last-level cache, we use the worst-case memory latency. In our evaluation, we found that the impact of flushing and writeback blocking is negligible.

While this work concentrates on timing channels through the NoC system, several other timing channels exist in real systems. We believe that the solution in this paper must be combined with a suite of side channel mitigation strategies (e.g., memory and cache partitioning) to address all conceivable system vulnerabilities.

2.4. Cloud Computing Threats

Because of resource sharing among various virtual machines, cloud computing is susceptible to side-channel attacks. Although EC2 Amazon [39] claims for isolation of distinct instances of virtual machines and it appears that they are running on individual resources, it utilizes the idea of virtualization. Therefore, it is conceivable that different instances of virtual machines use shared hardware resources. Therefore, elimination of side-channel attack is a significant area of concern.

2.5. Overview of the Proposed Approaches

Our proposed mapping algorithms are based on security levels or domains, which we categorize as L-shape isolated (Liso) and Isolated methods. We describe Liso-2, Liso-3, and Liso-4 for 2, 3 and 4 security levels, respectively. Also, we use *n* security levels for the Isolated method. In cloud computing, most of the owners provide 2, 3 or at most 4 security levels for their customers. In the worst case, we can assume that any customer demands for an isolated region, so we can provide any security levels by considering our approach.

The first approach maps each application individually as a rectangle shape. Each application is individually isolated and protected from others through LBDR algorithm. In fact, each router is independently provisioned with LBDR support, while deadlock-freedom and connectivity among all nodes are still guaranteed throughout execution. In addition, LBDR eliminates the need for additional hardware to recover from pathological scenarios such as deadlock. This is a great advantage in terms of silicon cost (and power), and limits the impact of reconfiguration on performance. Although XY-routing algorithm confines packets to the rectangle region, LBDR can make a tight border around the region of each application from any malicious attack.

Other three approaches are based on L-shape mapping. As shown in Figure 3, this kind of mapping starts mapping process of each application from a corner of the NoC based on its security level. It proceeds from corners of the NoC for each security level and stops when it meets other security levels. The basic idea is that the shape of each region remains as a rectangle with some penalty nodes. Also an LBDR implementation applies for each region boundary. The detailed explanation and objective of each approach is explained in Section 4.

In order to totally eliminate timing channels and provide non-interference, our approaches require resource allocation decisions to be independent from application demands. The main advantage of dynamic allocation is to gain performance in comparison to static allocation. Hence, this allocation will match the resource demands of the application. This efficient allocation can be satisfied by management software which can adjust the amount of resources needed by each demand. However, to maintain the security, the allocation decisions should not depend on confidential data. To solve this problem, the proposed approaches first allocate remaining nodes in each domain, and then allocate L-shape regions which are not necessarily equal to the demand, and therefore does not reflect sensitive, input-dependent data used at run-time. This mechanism is also right for Isolated approach, which assigns rectangles with some penalty nodes. In this case, we found that the network capacity can be dynamically allocated based on application demands without a security concern by giving unknown resources from previously freed nodes in that domain and L-shape pattern.

2.6. Problem Formulation

When dealing with possible region selection and mapping problem for the isolation of application with different security

Table 1: Summary of the Seven Conditions for HV-side Policy

HV-Side Policy		
Conditions name	Conditions	Mapping process
C1	X_length < Y_length&&RemAPP Size <= X_length	map horizontally up to X_length
C2	X_length < Y_length&&RemAPP Size > X_length&&RemAPP Size <= Y_length	map vertically up to Y_length
C3	X_length <= Y_length&&RemAPP Size > Y_length	map both horizontally and vertically
C4	X_length > Y_length&&RemAPP Size <= Y_length	map vertically up to Y_length
C5	X_length > Y length&&RemAPP Size <= X_length&&RemAPP Size > Y_length	map horizontally up to X length
C6	X_length > Y_length&&RemAPP Size > X_length	map both horizontally and vertically
C7	X_length == Y_length&&RemAPP Size <= X_length	
C7.1	H_dist <= V_dist	map horizontally up to X length

levels, we need to maximize the throughput of the system and at the same time, eliminate the interference for different domains. First, we need to define some terms for discussing our approaches. It should be noted that, we use the term rectangle for security level throughout the paper.

1. MD(n_1,n_2): Manhattan distance between node n_1(x_1, y_1) and n_2(x_2, y_2) where x_1, x_2, y_1, y_2 are x- and y- coordinates in the NoC, MD(n_1,n_2) = $|x_1 - x_2| + |y_1 - y_2|$.
2. L-size: The sum of height and width of a security level plus one.
3. SLn:CP(x, y): The coordinates of the outer corner point (vertex) of security level *n*.
4. X-length: The length of the side of the rectangle, which is parallel to the x axis.
5. Y-length: The length of the side of the rectangle, which is parallel to the y axis.
6. Domain: A set of applications that do not need to be isolated from each other. We also use security level for this term over the paper.
7. H-dist: The difference in x-coordinates of any left and right corner points of two security levels.
8. V-dist: The difference in y-coordinates of any top and bottom corner points of two security levels.
9. Penalty node: The node, which is available or unused in a security level.

In order to determine the type of mapping, we define four policies that are later used in Liso approach.

1. L-shape: In order to map an application, at first we calculate the L-size of a rectangular security level. Next, we subtract L-size from application size. If the result is positive we perform an L-shape mapping. This continues until a negative result is achieved. This way, the L-shape policy maps tasks to the nodes in the horizontal and vertical sides of a security level.
2. HV-side: In this policy, an application can be mapped if application size or the remaining tasks of the previous policy are less than L-size. Using this approach, HV-side policy makes a decision for mapping in one or both sides of a security level based on seven conditions, which are illustrated in Table 1. It maps in one side (horizontally or vertically) for five conditions of Table 1 (C1, C2, C4, C5 and C7) and maps in two sides for two conditions of Table 1 (C3 and C6).
3. H-side: Horizontal policy maps application tasks only on the side of a rectangle, which is parallel to the x-axis.
4. V-side: Vertical policy maps application tasks only on the side of a rectangle, which is parallel to the y-axis.

In order to enforce a security level to grow in a specific direction, we define three region-growing scenarios, which are used in Liso approach. It should be noted that, L-shape policy is the first policy in all the following scenarios. Default: This is the basic scenario and grows each security level diagonally. After deployment of L-shape policy, based on seven conditions of Table 1, it uses HV-side; otherwise, uses H-side or V-side policies.

1. Counter clockwise (ccw): This scenario tries to grow each security level in clockwise direction by means of H-side or V-side policy for each appropriate security level.
2. Horizontal-Vertical (HV): This scenario tries to grow two security levels in horizontal direction by means of H-side policy, and other two security levels in vertical direction by means of V-side policy.

3. Isolated Dynamic Mapping

In this section, we present our isolated dynamic mapping approaches. We describe the problem of region selection in order to find an appropriate region for each security level followed by a detailed description of isolated dynamic mapping approaches. Then, we propose the task allocation problem.

3.1. Region Selection Problem

The first step for each mapping approach is to find an appropriate region. When an application requests to map into the NoC, we should find a possible region for mapping. Based on the current configuration of the NoC and current state of each security level, our approaches should select an appropriate region for mapping. The region selection problem uses an appropriate region-growing scenario to solve the problem.

In the first mapping approach (Isolated), it finds a contiguous rectangle region closest to the control manager (CM) node with a size larger than or equal to the application size. It also results in minimum fragmentation for remaining nodes after

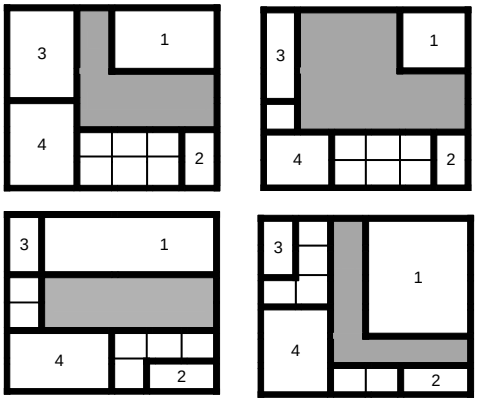


Figure 4: An example of region possible problem for security level 1 based on Table 2. (a) and (b) for possibility of 1, (c) for possibility of 2 and (d) for possibility of 3.

mapping. Then, this contiguous region is sent to the mapper for mapping process.

For the second approach (Liso-2), we have two security levels, which can grow from two different corners of the NoC. We observed that the best solution can be found when two security levels are placed on opposite corners. As stated, each security level forms a rectangle shape, and each rectangle has a corner point. Based on the position of the corner point of each rectangle, it is straightforward to calculate the remaining region for each security level.

Moreover, Liso-2 approach uses the *default* scenario to solve the region selection problem. Two regions can grow appropriately up to a prior limited partition. In fact, this partition divides the NoC into two identical regions. Now, based on the position of each region relative to this partition, one of the policies in the *default* region-growing scenario can be applied. The first policy in the *default* region-growing scenario is L-shape policy. According to this policy, each region grows diagonally. The second policy follows the HV-side policy. The third policy in this scenario applies H-side or V-side policy, which can be applied when the partition is met. Now, based on the current state of region, it can grow horizontally or vertically. In fact, this region selection problem grows each region conservatively, which results in high utilization.

The Liso-3 and Liso-4-default approaches follow the default region-growing scenario to solve the region selection problem. It should be noted that, before applying the default scenario, available regions should be decided based on the possibility conditions, which is illustrated in security level Tables 2, 3, 4 and 5 for regions 1, 2, 3 and 4, respectively. According to these possibility conditions, one of the possibilities can be happen. Liso-4-ccw approach accommodates the counter-clockwise scenario to solve the region selection problem. In this scenario, each region grows according to the L-shape policy; but when there is a choice between H-side and V-side policies, based on the position of each security level, it prefers the one that follows counter-clockwise direction.

Liso-4-HV approach works based on horizontal-vertical scenario to solve the region selection problem. At the first step, this scenario applies the *L-shape* policy for each security level region. Then, H-side or V-side policy can be applied by each region to follow the horizontal or vertical directions. More details for this approach is explained in Section Liso-3 and Liso-4.

Table 2: Possibility Conditions in Region Selection Problem for Security level 1

Security Level 1		
Possibility	Conditions	Available region
Poss.1	SL3.CP(x)>=SL4.CP(x) OR SL2.CP(y)>=SL4.CP(y)	SL3.CP(x), SL2.CP(y)
Poss. 2	SL4.CP(x)>=SL1.CP(x)	SL3.CP(x), SL4.CP(y)
Poss. 3	SL4.CP(y)>=SL1.CP(y)	SL4.CP(x), SL2.CP(y)

Table 3: Possibility Conditions in Region Selection Problem for Security level 2

Security Level 2		
Possibility	Conditions	Available region
Poss.1	SL4.CP(x)>=SL3.CP(x) OR SL3.CP(y)>=SL1.CP(y)	SL4.CP(x), SL1.CP(y)
Poss. 2	SL3.CP(x) >=SL2.CP(x)	SL4.CP(x), SL3.CP(y)
Poss. 3	SL2.CP(y) >= SL3.CP(y)	SL3.CP(x), SL1.CP(y)

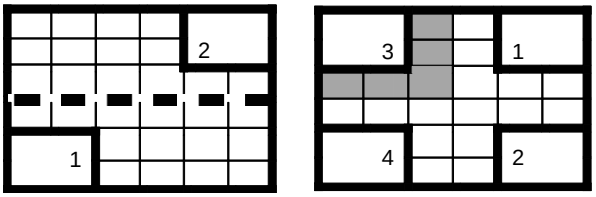
Table 4: Possibility Conditions in Region Selection Problem for Security Level 3

Security Level 1		
Possibility	Conditions	Available region
Poss.1	SL2.CP(x)>= SL1.CP(x) OR SL4.CP(y) >=SL2.CP(y)	SL1.CP(x), SL4.CP(y)
Poss. 2	SL3.CP(x)>= SL2.CP(x)	SL1.CP(x), SL2.CP(y)
Poss. 3	SL2.CP(y) >= SL3.CP(y)	SL2.CP(x), SL4.CP(y)

Table 5: Possibility Conditions in Region Selection Problem for Security Level 4

Security Level 1		
Possibility	Conditions	Available region
Poss.1	SL1.CP(x)>= SL2.CP(x) OR SL1.CP(y)>= SL3.CP(y)	SL2.CP(x), SL3.CP(y)
Poss. 2	SL4.CP(x) >= SL1.CP(x)	SL2.CP(x), SL1.CP(y)
Poss. 3	SL4.CP(y) >= SL1.CP(y)	SL1.CP(x), SL3.CP(y)

An illustrative example is shown in Figure 4; here we assume that an application has the security level of 1. Therefore, we consider conditions of Table 2 for describing this example. In Figure 4, the highlighted squares stand for the nodes, which form possible regions for mapping the next application into security level 1. In other words, highlighted region is a maximum bound area which is surrounded by other security levels. Figure 4a and Figure 4b show the states of possible regions for conditions that are true for the first possibility of Table 2. In Figure 4a, the first condition of possibility 1 in Table 2 means that the x coordinate of corner point of security level 3 is greater than, or equal to the x coordinate of corner point of security level 4. Therefore, the available growing-region for security level 1 binds to the x coordinate of security level 3 from the left side, and binds to the y coordinate of security level 2 from the bottom side. In Figure 4b, the growing-region for the security level 1 is bound to the y coordinate of security level 2 from bottom, because its y coordinate is greater or equal to the y coordinate of security level 4, and is bound from the left side to the x coordinate of security level 3. The states of regions for other possibilities are shown in Figure 4c and Figure 4d.



(a) Logical border line (b) Security level L-size
Figure 5: Characteristics of NoC in Liso approach.

3.2. Isolated Approach

The best found contiguous rectangle from region selection method are passed to the mapping algorithm. Afterward, in the task allocation step, application tasks are mapped inside the contiguous and completely isolated rectangle area. For our problem, we assume that our system can support *n* security levels. Any user may claim for a separate region in the system for executing his/her program. Therefore, we provide *n* domains for hosting *n* users. Our rectangle area for mapping is sufficient for isolation, because XY-routing prevents packets from exiting the region. Moreover, LBDR mechanism is applied in order to account for the tight constraints regarding malicious attacks from other applications. LBDR allows uninterrupted operation even during reconfiguration. This choice reduces the signaling between NoC switches and control manager, since configuration of the routing logic depends on just few configuration bits. More details of this mechanism can be found in [34].

3.3. L-shape Isolated Approach

Now, we describe our L-shape isolated approaches. First, we present L-shape mapping in Liso-2 approach for two security levels. Next, we describe L-shape mapping in Liso-3 and Liso-4 approaches for three and four security levels, respectively.

3.3.1 Liso-2

Liso-2 approach works for two security levels as shown in Figure 5. In this approach, we assume that our NoC configuration accepts applications for only two security levels; thus, they can map from two corners of the network. When an application requests for entering the NoC, at first the control manager starts mapping according to: 1) application characteristics; 2) security level of the application; and 3) current NoC configuration.

In this approach, at first, applications for each security level are mapped based on L-shape policy until one of them meets the prior border constraint of the NoC as shown in Figure 5a with dashed line. This logical border line breaks down the total NoC area into two regions, from height or width. If one of the security levels meets this logical border, then mechanism of mapping for both security levels changes its policy from L-shape to H-side or V-side policy. The mapping process proceeds with this new policy to the point where the configuration of the NoC returns to the state of unmet border. The steps of Liso-2 algorithm are illustrated in Algorithm 1 (Logical border is assumed to divide the NoC horizontally).

Appertain to this approach our objective is to maximize the expected utilization and throughput of the system.

3.3.2 Liso-3 and Liso-4

For Liso-3 and Liso-4, we do not set any border constraint in the NoC; therefore, there may exist more than one solution, increasing the utilization of the system. To grow security levels from each corner of the NoC, we use two additional scenarios (counter clockwise and horizontal-vertical) for region-growing. In this case, we only consider these scenarios for Liso-4. At first, we discuss default scenario, which is used for both approaches. In all scenarios, as soon as an application enters the system, we calculate its size (T-size) and compare it with L-size. As shown in Figure 5b, L-size is the size of the border nodes for each current configuration of a security level in the NoC. For example, L-size value for security level 3 is equal to 5. Then, for application size of more than or equal to L-size, it first uses L-shape policy, and then each scenario decides based on its policies.

We use three policies for each scenario. The first policy is named HV-sides, and it means that there are available nodes on both horizontal and vertical sides. Therefore, based on the number of available nodes on both sides and application tasks, one of seven conditions will be matched.

Algorithm 1 Liso-2's algorithm

Input: Task Graph *AG* (*T* , *E*, *S*) of application *Ap*
and current possible region(*R*) in NoC.
Output: map *R* ← *T*
NoC size: ← *M* × *N*
borderline ← Divide height or width of the NoC
into two areas.
while borderline is unmet **do**
  if T-size ≥ L-size **then**
    for each task *ti* ∈ *T* **do**
      L-shape(*ti* , *R*)
    end for
  else
    for each task *ti* ∈ *T* **do**
      HV-side(*ti* , *R*)
    end for
  end if
end while
{borderline is met}
if X-length(*S*) < *M* **then**
  for each task *ti* ∈ *T* **do**
    V-side(*ti* , *R*)
  end for
else if *S.CP* (*y*) < *Sothersecuritylevel .CP* (*y*)) **then**
  for each task *ti* ∈ *T* **do**
    H-side(*ti* , *R*)
  end for
end if

Table 6: Summary of the Order of Applying Policies for the L-shape Approaches

Approach	First Policy	Second Policy	Third Policy	Forth Policy
Liso-2	L-shape	HV-side	H-side or V-side	
Liso-3	L-shape	HV-side	H-side or V-side	
Liso-4-default	L-shape	HV-side	H-side or V-side	
Liso-4-ccw	L-shape	H-side(2, 3), V-side(1, 4)	HV-side	H-side(1, 4), V-side(2, 3)
Liso-4-HV	L-shape	H-side(3, 4), V-side(1, 2)	HV-side	H-side(1, 2), V-side(3, 4)

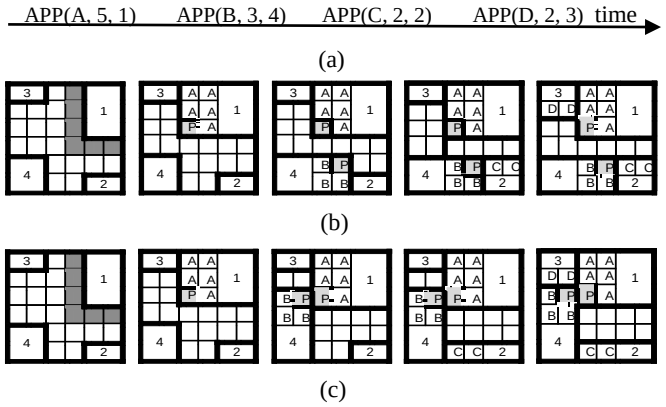


Figure 6: Example showing the counter clockwise and Horizontal-Vertical scenarios for region growing on four applications: (a) Time of arriving applications. (b) Liso-4-ccw: It applies V-side policy for the first two entered applications (A and B) and then applies H-side policy for C and D applications. (c) Liso-4-HV: It applies V-side policy for applications with security levels of 1 and 2 (application A and C) and H-side policy for applications with security levels of 3 and 4 (application B and D).

These seven conditions are shown in Table 1. Second and third policies are met if only there are available nodes on one side (horizontal or vertical side). These policies are named H-side and V-side. Default scenario works according to these three policies as mentioned above. The steps of Liso-4 algorithm are illustrated in Algorithm 2.

Counter clockwise scenario grows each security level in a counter clockwise manner. It assumes that security levels 1, 2, 3 and 4 are located at the right-up, right-down, left-up and left-down of the NoC, respectively. Each security level, can grow towards the other security levels by taking either H-side or V-side policies. An illustrative example is shown in Figure 6. Here, we assume that four applications are entered to the system. At first, by entering the "A" application with size of 5, which belonging to the security level 1, because its size is less than L-size (6) it cannot use L-shape policy. Therefore, according to counter clockwise scenario, it applies V-side policy if there are enough nodes along y-axis toward security level of 3, otherwise it applies HV-side or H-side policy. Next, application "B" with size 3 and security level 4 is entered. This also uses V-side policy. As shown in Figure 6b, for the remaining applications, the H-side policy is used.

Horizontal-vertical scenario grows the region of security levels 1 and 2 horizontally and grows the region of security levels 3 and 4 vertically (See Figure 6c). In this case, the security levels 1 and 2 use the V-side policy in order to grow horizontally. The security levels 3 and 4 grow vertically through applying H-side policy. Table 6 summarizes the order of accomplishing policies for each approach.

Algorithm 2 Liso-4's algorithm

Input: Task Graph *AG* (*T* , *E*, *S*) of application *Ap*
and current possible region(*R*) in NoC.
Output: map $R \leftarrow T$
if T-size ≥ L-size **then**
  for each task $ti \in T$ **do**
    L-shape(t_i , *R*)
  end for
else if S.CP(*x*) < R.CP(*x*) **and** S.CP(*y*) < R.CP(*y*) **then**
  for each task $t_i \in T$ **do**
    HV-side(t_i , *R*)
  end for
else if S.CP(*x*) ≥ R.CP(*x*) **and** S.CP(*y*) < R.CP(*y*) **then**
  for each task $t_i \in T$ **do**
    H-side(*ti* , *R*)
  end for
else if S.CP(*x*) < R.CP(*x*) **and** S.CP(*y*) ≥ R.CP(*y*) **then**
  for each task $t_i \in T$ **do**
    V-side(t_i , *R*)
  end for
end if

3.4. Task Allocation

After the region selection process finds an appropriate region, we continue allocating the tasks of incoming applications to the cores in the isolated region while minimizing the interprocessor communication. The selection of the first core is important for starting task allocation [38]. For the sake of simplicity, we find the core with the most neighbors in the selected region as the first core. By default, either the core at the corner point or the last core in the selected region can be considered as the first candidate core. To indicate the state of each task in the task allocation process, we specify each task with A-level, B-level or C-level. At first, all tasks are A-level, and may later become either B-level or become directly C-level. Also, a B-level task is declared as a node which is mapped onto a tentative core, which may change later. A core is set to be unavailable after a C-level task is mapped onto it. We define two instructions for tasks.

1. **LOCATE**. This consists of the following: 1) Select available cores with most neighbors in the selected region for task t and 2) define task t as B-level; task t considered as "LOCATE".

2. **BIND**. This consists of the following: 1) Select a specific core for task t, such that the distance between task t and its C-level neighboring tasks is minimized; if more than one core gets the minimum distance, we select the core

that has the closer number of unutilized neighbors to the number of A-level or B-level neighbors of task t and 2) define task t to C-level; then task t is considered as "BIND".

We first sort tasks into an ordered set using the nonincreasing order of their total communication volume. In other words, a task with higher communication volume can locate or bind earlier. The task allocation algorithm is summarized in Algorithm 3.

Algorithm 3 Task allocation algorithm

Input: Ordered set of tasks in task graph
AG (*T* , *E, S*) of application *Ap* , chosen first core and current selected region(*R*) in NoC.
Output: All tasks level changes to C-level
(allocate $R \leftarrow T$).
for each task $ti \in T$ **do**
    Level (*ti*) = *A* − *level*
end for
$t \leftarrow$ first smallest A-level task in ordered set.
while there exists any A- or B-level tasks in the ordered set **do**
    if neighbors of $t \neq$ B- or C-level **then**
        LOCATE(*ti*)
    end if
    if neighbors of *t* = B- or C-level **then**
        BIND(*ti*)
    end if
end while

4. Evaluation

In this section we first describe our platform and application model. Then, we define evaluation metrics for experimental setup. We conclude our evaluation with the latency, power consumption and throughput analysis.

4.1. Platform Description

Our NoC topology, T(N, L), consists of identical cores integrated by a 2-D *n*×*n* mesh network, which is managed by a master core, control manager (CM), residing in $n_{x.y} = (0.0)$. The NoC topology (*T*) contains a set of nodes $n_{x.y} \in N$, connected together through communication links $l_k \in L$. A task t_i can be assigned to a node, denoted by n_{t_i} . The CM executes our proposed mapping algorithm, and allocates appropriate resources to a newly arrived application. When a task is assigned to a slave core, it starts executing it. Once the application finishes its execution and leaves the system, the slave cores send their addresses to the manager and notify it that they have become available. The communication infrastructure consists of channels, which connect router of each node together via a standard network interface. In terms of functionality, the network is designed for transmitting packets under XY-routing algorithm and wormhole switching.

4.2. Application Model and Characteristics

An application is a directed graph G (V, E) with vertices V and edges E. Each vertex v_i in V symbolizes one task of the application A(p). Each edge $e_{i.j}$ in E represents the communication value ($w_{i.j}$) from vertex $v_i$to vertex v_j.

4.3. Evaluation Metrics

Our basic metrics for system performance are evaluation of latency, in clock cycles, of the network and impact on throughput of the system. We also evaluated power consumption overhead in terms of average weighted manhattan distance (AWMD) [39]. AWMD, as a metric to evaluate the power consumption of a mapped application, is the summation of weighted manhattan distances (MD), as defined by Equation 1:

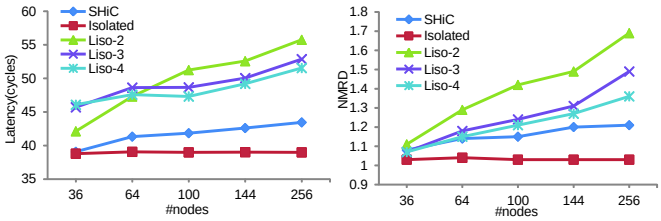
$$\boldsymbol{AWMD_{map(A_p)}} = \frac{\sum_{\forall e_{i.j} \in E} \boldsymbol{w_{i.j}} \times \boldsymbol{MD_{n_{t_i}.n_{t_j}}}}{\sum \boldsymbol{w_{i.j}}}. \qquad (\mathbf{1})$$

We used normalized mapped region dispersion (NMRD) [38] value as a metric for evaluating the squareness of the mapped application. In other words, NMRD can be used to show how much an application is fragmented, where the NMRD value of 1 means a squared region. Chou et al. [36] proposed L_1 distance as metric to show the sum of pairwise manhattan distances between all locations within a circular region. This metric modified as MRD in [38] for a squared area with |T| nodes. In fact, this factor is highly related to the performance of an application and network because contiguous mapping of an application can impact internal and external congestions. NMRD is defined by Equation 2:

$$\boldsymbol{NMRD_{map(A_p)}} = \frac{\boldsymbol{MRD_{map(A_p)}} - \boldsymbol{MRD_{SQ(|T|)}}}{\boldsymbol{MRD_{SQ(|T|)}}}. \qquad (\mathbf{2})$$

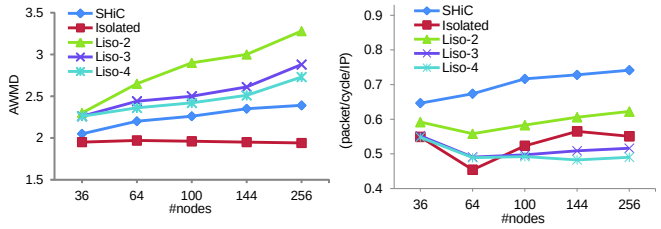
4.4. Experimental Setup

The experiments are performed on an Intel Core™ 2 CPU T7400 running at 2.16 GHz and 2.17 GHz. We modified Noxim [40], a cycle-accurate SystemC many-core platform, to evaluate performance of our proposed isolation mapping algorithms. The NoC topology size is configured from 6 × 6 to 20 × 20 nodes. Several sets of workloads of random applications are generated by TGFF [41], where the communication volumes are randomly distributed between 2 to 16 flits of data. We used 2 virtual channels, each having an eight-flit buffer. A random sequence of applications is entered into the scheduler FCFS according to the desired rate. The sequence is kept fixed in all experiments for the sake of fair comparison.



(a) Average network latency (b) NMRD

Figure 7: The effect of dispersal vs. contiguous mapping on latency for different network sizes.



(a) AWMD: As a power metric (b) Throughput

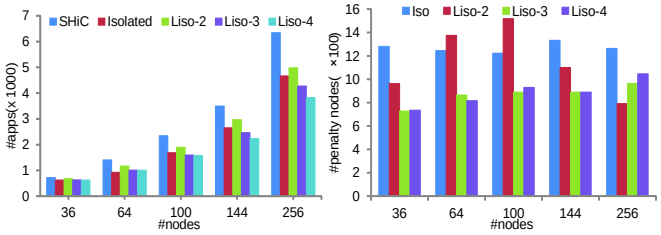
Figure 8: Power consumption and throughput for different number of applications and nodes network.

4.5. Simulation Results

In this section, we report the simulation results for latency, power consumption and throughput of our schemes. The objective of our experiments is to evaluate the throughput and efficiency of our approaches towards on-chip network security, specially for timing channels protections.

Latency. We first examined the impact of our schemes on average latency with different numbers of applications, and different numbers of nodes. In order to clearly understand the effect of security level and how the overhead scales with network sizes, we measured latency for different network sizes in Figure 7a. In this figure, we plotted latency in cycles (y-axis) vs. number of nodes on the x-axis for network sizes varying from 36 to 256 nodes with 512 applications. It is clear that the latencies of Liso schemes and SHiC increase with network size due to a dispersed mapped area. For example, the overhead is increased from 39.06 (43.43) to 42.11 (55.74) cycles by 7.8% (28.3%) for network size of 36 (256) nodes with 512 applications. We can see that the latency of Isolated scheme is almost independent of the network size, leading to a line with constant value. On the other hand, the larger the network, the higher is the overhead for Liso-2 because more dispersed mapping leads to more external congestion. This clearly shows that Isolated scheme is scalable with network size. In order to clearly understand why the overhead of Liso's schemes increases with network sizes, we plotted NMRD as the dispersion metric in Figure 7b. As it can be seen, the dispersion value of Liso-2 increases due to a higher fragmented mapping. This is also true for both Liso-3 and Liso-4.

Power Consumption. We evaluated the power consumption overhead for our schemes. The power consumptions are extracted from AWMD metric. The extracted power values follow the same trend as AWMD. The results are plotted in



(a) Utilization (b)Penalty nodes

Figure 9: Performance impact of different isolation mapping approaches for different number of nodes.

Figure 8a for different numbers of network sizes with 512 applications. Figure 8a clearly shows that Liso-2 performs poorly since there are many non-contiguous regions in the system. We showed that for network sizes of 36 nodes, the Isolated approach achieves about 5% and 17% energy savings, compared to SHiC and Liso-2 approaches, respectively. The saving is even greater (up to 69%) for a 256-node network. As it can be seen, power consumptions of other approaches follow almost the same trend and increase with network size, while Isolated keeps its preeminence over the other approaches.

Throughput and Utilization. Throughput is a first order concern for concurrent many-core systems. An efficient dynamic mapping needs to guarantee the resulting mapping at run-time. In terms of performance metric, we evaluated the effect of non-interference and isolation on throughput regarding different levels of security. We ran a set of experiments for millions of cycles where thousands of applications enter and leave the system. An application is pushed into the scheduler FIFO as soon as there are enough resources available for mapping the application into the specified security level. This is to achieve the maximum possible system throughput. Accordingly, applications experience waiting times according to their policies. Note that the turnaround time of an application is calculated from the moment it is pushed into the scheduler FIFO.

We observed that throughput (utilization) improves by 14% (14%) on average for Liso-2 compared to the Isolated approach, as shown in Figure 8b (Figure 9a). These gains are made possible due to relax allocation of different resources for a shared security level, leading to increased resource utilization, as opposed to the Isolated approach, which confines each application individually to make more security levels. In order to clarify the effect of penalty nodes and fragmentation after each mapping on utilization of the proposed approaches, we plotted networks of sizes 36 and 256 nodes with 512 applications in Figure 9b. As it can be seen, penalty nodes for Liso-2, Liso-3 and Liso-4 decreases 9%, 32% and 30% on average in comparison to Isolated approach. However, Liso-3 and Liso-4 cannot gain performance over Isolated approach. This is due to the fact that by increasing the number of security levels, by entering an application with one security level, there are some free nodes in other security levels and the waiting time for each region rises while in Isolated approach the applications do not experience such waiting time.

In order to show how dynamic isolation is better than static isolation, we also compared utilization of dynamic isolation approaches with static allocation. We set up 4 identical regions for 4 security levels for Liso-4 approach. It means that based

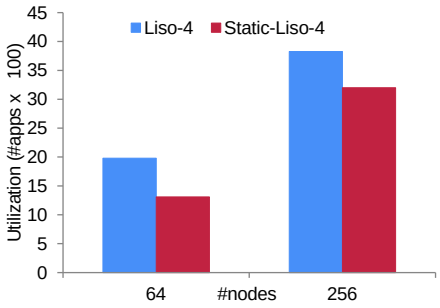


Figure 10: Performance impact of static isolation vs. dynamic isolation for different number of nodes.

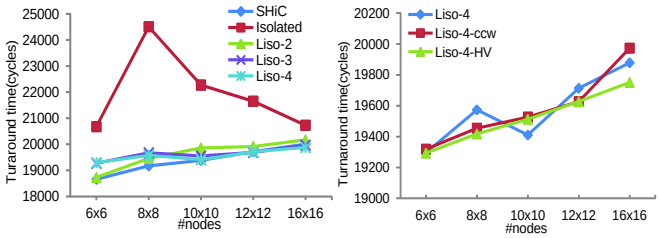
on network size, each security level is confined evenly. For example, for a network size of 64 we assume that each security level has 16 nodes. It should be noted that for network of size 64, we consider applications with at most 16 tasks. We run Liso-4 and Static-Liso-4 for 10,000,000 cycles for network size of 64 and 256 nodes. It can be seen that utilization improves by 34% and 16% for network of sizes 64 and 256 for dynamic approach compared to the static approach, as shown in Figure 10.

In order to clarify how the turnaround time of the Isolated approach scales with network size, we plotted networks of varying sizes from 36 to 256 nodes with 512 applications in Figure 11a. As it can be seen, baseline and Liso's schemes outperform the Isolated approach, but by increasing the size of network dispersion mapping and consequently, the increased execution time ultimately result in more turnaround time of applications. This shows that our Isolated approach looks promising for large NoC platforms.

We also evaluated turnaround time of two alternative scenarios (ccw and HV) for Liso-4 approach to test the benefits of them. Figure 11b shows the results for network sizes varying from 36 to 256 nodes with 512 applications. Our experiments show that HV scenario is more stable than ccw scenario when the size of network increases. It specially retains better turnaround time when the size of network is 256-nodes. HV scenario provides better resource allocation due to growing security levels in more effective regular regions. Thus, turnaround time improves by 30% and 27% on average for HV scenario, compared to the ccw and default scenarios, respectively.

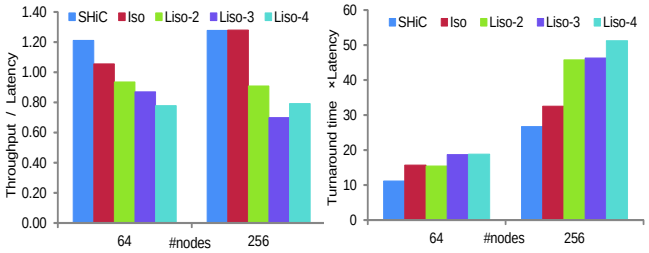
Liso-approach can support – by construction – 2, 3 and 4 domains; explicit mapping provides the number of active domains appropriately. However, at the network level, and depending on whether the number of domains is required to the identified NoC constraints, increasing the number of domains may lead to unsatisfactory performance. When we cannot align the number of domains to match the conditions required for better performance than isolated approach, we can follow isolated approach, which enables the mapping isolation of a large number of domains with a controllable latency overhead.

To evaluate the effect of the number of security levels (domains) on performance we conducted an additional experiment. Figure 7a and Figure 8b show the latency and throughput of Isolated and Liso's approaches when varying the size of networks. In terms of latency, Liso-3 and Liso-4 approaches show better performance, both outperform Liso-2 approach. On the other hand, Isolated can absorb the overhead



(a) (b)

Figure 11: Performance impact of different isolation mapping approaches for different number of nodes. (a) Turnaround time. (b) Turnaround time for two extended scenarios in Liso-4. HV scenario leads to better turnaround time compared to the ccw and default (27% and 30% on average, respectively).



(a) (b)

Figure 12: Performance impact of different isolation mapping approaches for different number of nodes.
(a) Throughput/Latency. (b) Turnaround time × Latency.

due to contiguous mapping. Figure 8b shows the throughput of different approaches, where both Isolated and Liso-2 approaches perform better in comparison with Liso-3 and Liso-4.

To capture the interaction between latency and throughput, we employed the combined metric of (Throughput/Latency). Under this metric (higher is better), both Isolated and Liso-2 exhibit better performance as shown in Figure 12a. Specifically, Isolated outperforms Liso-3 and Liso-4 by 21% (83%) and 36% (62%) for network size of 64 (256) nodes, respectively, in terms of this combined metric. Again, we employed the combined metric of (Turnaround time × Latency). Based on this metric (lower is better), both Liso-2 and Isolated have better performance as shown in Figure 12b. Note that the support of additional domains through Liso approaches implies additional performance overhead, due to delay and congestion. Hence, Isolated with low latency compared to Liso-3 and Liso-4 can be considered as the best candidate to support additional domains.

We show the isolation properties of Liso approach through evaluation of the accepted throughput of two security levels. In this case, we run the Liso-2 approach, which involves a two-domain network with size of 64 nodes, three times. We categorize 32 applications into two sets of 16 applications for security levels 1 and 2.

We evaluated the throughput of each security level for two configurations. In the first configuration, we evaluated the throughput of each security level while executing Liso-2 separately for 16 applications with specified security level. In the second configuration, we evaluated the throughput of each security level while executing Liso-2 approach for 32

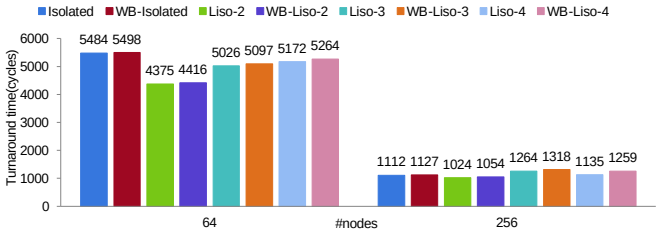
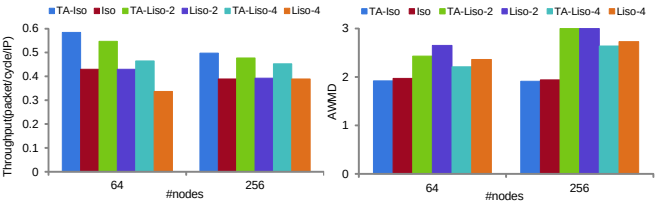


Figure 13: Performance overhead of context switching for non-interference via core mapping.



(a) Throughput (b) Power consumption

Figure 14: Performance impact of task allocation for different number of nodes.

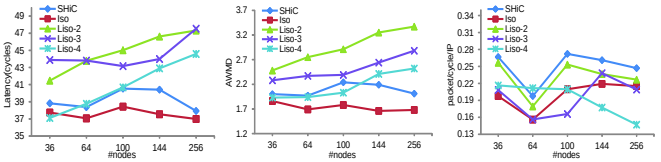
applications with security level 1 and 2. We observed that the throughputs of security levels for the two configurations are equal. It means that the load in one security level is completely isolated from the changes in other security levels, and does not alter the accepted throughput.

In order to support non-interference via core mapping, we evaluated turnaround time of each proposed approach with flushing. Figure 13 shows the turnaround time of Isolated, Liso-2, Liso-3 and Liso-4 with and without flushing. The overhead of context switching is quite small. On average, the overhead for network size of 64 (256) is 1% (5%).

We evaluated the throughput and power consumption of our approaches with the applied online task allocation algorithm. As shown in Figure 14a, we observe that for TA-Isolated, TA-Liso-2 and TA-Liso-4 approaches about 36\% (28%), 27% (22%) and 38% (16%) performance gains are achieved respectively, compared to Isolated, Liso-2 and Liso-4 approaches for 64 (256) network nodes. We also observe that about 6% (3%) power savings can be achieved, on average, compared to Isolated, Liso-2 and Liso-4 approaches for 64 (256) network nodes, respectively (see Figure 14b).

Real Applications Result. As a realistic benchmark to evaluate the quality of the approaches, we opted H.264 decoder, Unmanned Air vehicle (UAV) application and the decoder of video object planes, referred as VOPD application. The H.264 standard has been used in a wide range of applications such as mobile phone, tablet, Blueray Disc, YouTube, broadcast, and real-time video conference. VOPD implementation is usually computationally intensive, making it a good candidate for NoC-based systems. These application models are considered as our benchmarks.

The following results show the potential of our proposed isolated mapping algorithms in real applications. We first examined the impact of our schemes on average latency in Figure 15a. In this figure, we plotted latency in cycles (y-axis) vs. number of nodes on the x-axis for network sizes varying from 36 to 256 nodes with 512 applications. It is clear that the



(a) Latency (b) Power consumption (c) Throughput

Figure 15: Performance impact of different isolation mapping for different number of nodes.

latencies of Liso schemes increase with network size due to a dispersed mapped area. For example, the overhead of Liso-2 is increased from 38.82 (37.93) to 41.47 (47.32) cycles by 7% (25%) for network size of 36 (256) nodes with 512 applications. It can be seen that the latency of Isolated scheme is almost independent of the network size, leading to a line with constant value.

We evaluated the power consumption overhead for our schemes. The results are plotted in Figure 15b for different numbers of network sizes with 512 applications. Figure 15b clearly shows that Liso-2 performs poorly, since there are many non-contiguous regions in the system. We showed that for network sizes of 36 nodes, the Isolated approach achieves about 7% and 25% energy savings, compared to SHiC and Liso-2 approaches, respectively. The saving is even greater (up to 50%) for a 256-node network.

We observed that throughput is improved by 16% on average for Liso-2 compared to the Isolated approach, as shown in Figure 15c.

5. Conclusion

In this paper, a dynamic mapping strategy for constructing high-throughput isolation has been developed. This mapping eliminates interference among applications for NoC resources. Thus, it offers high performance and no information leakage under various scenarios. The goal of our approaches is to provide more security for applications with common communication resources. The main idea is to have dynamic isolated regions across a NoC, for applications with different security levels, and map applications to these regions effectively. Within each security region, applications are mapped efficiently to improve system throughput. Isolated and Liso yield in performance degradations of 25.08% and 15.56%, respectively, relative to a non-secure baseline. Liso-2, Liso-3 and Liso-4 approaches out-perform Isolated by 14%, 2.1% and 1.51%, respectively. The main idea of the proposed approaches is efficient mapping of applications while avoiding side channel and interference. Unlike the previous techniques, Liso and Isolated has provided security under efficient and cost-effective DOR routing mechanism, and thus has yielded in higher performance.

We conclude that our isolation methodology for NoCs provides high system and application throughput (performance), and hope that the proposed dynamic isolation mapping schemes can provide a simple framework for designing secure on-chip networks, which are applicable in many-core cloud and embedded systems.

References

[1] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in Proc. 34th Int. Symp. Computer Architecture, pp. 494–505, 2007.

[2] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in Proc. 41th IEEE/ACM Int. Symp. Microarchitecture, pp. 83–93, 2008.

[3] Z. Wang, "Information Leakage Due to Cache and Processor Architectures," Princeton Univ., 2012.

[4] D. J. Bernstein, "Cache-timing attacks on AES." 2005.

[5] Y. Wang, A. Ferraiuolo, and G. E. Suh, "Timing channel protection for a shared memory controller," in Proc. 20th IEEE Int. Symp. High Performance Computer Architecture, pp. 225–236, 2014.

[6] A. Gundu et al., "Memory bandwidth reservation in the cloud to avoid information leakage in the memory controller," in Proc. 3rd Workshop Hardware Architectural Support Security Privacy, pp. 1–5, 2014.

[7] B. Grot et al., "Memory bandwidth reservation in the cloud to avoid information leakage in the memory controller," in Proc. 3rd Workshop Hardware Architectural Support Security Privacy, vol. 39, no. 3, pp. 199–212, 2010.

[8] A. Shafiee, A. Gundu, M. Shevgoor, R. Balasubramonian, and M. Tiwari, "Avoiding Information Leakage in the Memory Controller with Fixed Service Policies," in Proc. 48th ACM Int. Symp. Microarchitecture, pp. 89–101, 2015.

[9] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in Proc. 6th IEEE/ACM Int. Symp. Networks on Chip, pp. 142–151, 2012.

[10] H. M. G. Wassel et al., "SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip," in Proc. 40th Int. Symp. Computer Architecture, pp. 583–594, 2013.

[11] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in Proc. 16th ACM conf. Computer communications security, pp. 199–212, 2009.

[12] N. R. Potlapally, A. Raghunathan, S. Ravi, N. K. Jha, and R. B. Lee, "Satisfiability-based framework for enabling side-channel attacks on cryptographic software," in Proc. conf. Design, automation test Europe, pp. 18–23, 2006.

[13] P. Barham et al., "Xen and the art of virtualization," SIGOPS Oper. Syst. Rev., vol. 37, no. 5, pp. 164–177, Oct. 2003.

[14] A. Velte and T. Velte, Microsoft virtualization with Hyper-V. McGraw-Hill, Inc., 2009.

[15] B. Walters, "VMware virtual platform," Linux J., vol. 1999, no. 63es, Jul. 1999.

[16] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," in Proc. 19th ACM conf. Computer communications security, pp. 305–316, 2012.

[17] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based defenses against cross-vm side-channels," in Proc. 23th USENIX Conf. Security Symp., pp. 687–702, 2014.

[18] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: A template for composable and predictable multi-processor system on chips," ACM Trans. Des. Autom. Electron. Syst., vol. 14, no. 1, pp. 1–24, Jan. 2009.

[19] F. Triviño, J. L. Sánchez, F. J. Alfaro, and J. Flich, "Network-on-chip virtualization in chip-multiprocessor systems," J. Syst. Arch., vol. 58, no. 3, pp. 126–139, Mar. 2012.

[20] Å. G. Solheim, O. Lysne, T. Sødring, T. Skeie, and J. A. Libak, "Routing-contained virtualization based on Up*/Down* forwarding," in Proc. 14th Int. Conf. High Performance Computing, pp. 500–513, 2007.

[21] A. Mejia, J. Flich, and J. Duato, "On the potentials of segment-based routing for NoCs," in Proc. 37th IEEE Int. Conf. Parallel Processing, pp. 594–603, 2008.

[22] R. Schooler, "Tile processors: Many-core for embedded and cloud computing," in Proc. 14th IEEE Workshop High Performance Embedded Computing, 2010.

[23] B. Grot, S. W. Keckler, and O. Mutlu, "Preemptive virtual clock: a flexible, efficient, and cost-effective QOS scheme for networks-on-chip," in Proc. 42nd IEEE/ACM Int. Symp. Microarchitecture, pp. 268–279, 2009.

[24] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees," SIGARCH Comput. Arch. News, vol. 39, no. 3, pp. 401–412, Jun. 2011.

[25] P. Lotfi-Kamran, B. Grot, and B. Falsafi, "NOC-Out: Microarchitecting a scale-out processor," in Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 177–187, 2012.

[26] J. W. Lee, M. C. Ng, and K. Asanovic, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in Proc. 35th Int. Symp. Computer Architecture, pp. 89–100, 2008.

[27] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," IEEE Des. Test Comput., vol. 22, no. 5, pp. 414–421, Sep. 2005.

[28] R. Stefan and K. Goossens, "Enhancing the security of time-division-multiplexing networks-on-chip through the use of multipath routing," in Proc. 4th ACM Int. Workshop Network on Chip Architectures, pp. 57–62, 2011.

[29] L. Chen, K. Hwang, and T. M. Pinkston, "RAIR: Interference Reduction in Regionalized Networks-on-Chip," in Proc. 27th IEEE Int. Symp. Parallel Distributed Processing, pp. 153–164, 2013.

[30] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A Hardware Design Language for Timing-Sensitive Information-Flow Security," in Proc. 20th Int. Conf. Architectural Support Programming Languages Operating Systems, pp. 503–516, 2015.

[31] E. Papastefanakis, B. Maitre, and D. Ragot, "Security Challenges in ManyCore Embedded Systems based on Networks-on-Chip (NoCs)," in Proc. Workshop Embedded Systems Security, pp. 1–6, 2015.

[32] L. Fiorin, G. Palermo, and C. Silvano, "MPSoCs run-time monitoring through Networks-on-Chip," in Proc. Conf. Design, Automation Test Europe, pp. 558–561, 2009.

[33] A. Motakis, G. Kornaros, and M. Coppola, "Dynamic resource management in modern multicore SoCs by exposing NoC services," in Proc. 6th IEEE Int. Workshop

Reconfigurable Communication-centric Systems-on-Chip, pp. 1–7, 2011.
[34] J. Flich et al., "On the potential of NoC virtualization for multicore chips," in Proc. IEEE Int. Conf. Complex, Intelligent Software Intensive Systems, pp. 801–807, 2008.
[35] H. Lu, G. Yan, Y. Han, B. Fu, and X. Li, "RISO: relaxed network-on-chip isolation for cloud processors," in Proc. 50th Annu. Design Automation Conf., pp. 38:1--38:6, 2013.
[36] C.-L. Chou and R. Marculescu, "Run-Time Task Allocation Considering User Behavior in Embedded Multiprocessor Networks-on-Chip," IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 29, no. 1, pp. 78–91, Jan. 2010.
[37] M. Fattah et al., "Mixed-criticality run-time task mapping for noc-based many-core systems," in 22nd Euromicro Int. Conf. Parallel, Distributed and Network-Based Processing ,pp. 458–465, 2014.
[38] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart Hill Climbing for Agile Dynamic Mapping in Many-core Systems," in Proc. 50th ACM Design Automation Conf. , pp. 39:1--39:6, 2013.
[39]"Amazon EC2." [Online]. Available: https://aws.amazon.com/ec2/.
[40] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in IEEE 26th Int. Conf. Application-specific Systems, Architectures Processors, pp. 162–163, 2015.
[41] K. Vallerio, "Task graphs for free (TGFF v3. 0)." 2008.

Mohammad Sadegh Sadeghi received the M.S. degree in computer engineering from Amirkabir University of Technology (Tehran Polytechnic), in 2007, the Ph.D. degree in computer engineering in the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. His research focuses on security for large-scale system on chip, with a special emphasis on run-time optimization for timing channel protection for on-chip networks.

Email: mssadeghi@ce.sharif.edu

Siavash Bayat Sarmadi received the B.Sc. degree from the University of Tehran, Iran, in 2000, the M.Sc. degree from Sharif University of Technology, Tehran, Iran, in 2002, and the PhD degree from the University of Waterloo in 2007, all in computer engineering (hardware). He was with Advanced Micro Devices, Inc. for about 6 years. Since September 2013, he has been a faculty member in the Department of Computer Engineering, Sharif University of Technology. He has served on the executive committees of several conferences. His research interests include hardware security and trust, cryptographic computations, and secure, efficient and dependable computing and architectures. He is a member of the IEEE.

Email: sbayat@sharif.edu

Shaahin Hessabi received the B.Sc. and M.Sc. degrees in Electrical Engineering from Sharif University of Technology, Tehran, Iran in 1986 and 1990, respectively. He received his Ph.D. degree in Electrical and Computer Engineering from University of Waterloo, Waterloo, Ontario, Canada in 1995. He joined Sharif University of Technology in 1996, and is currently an associate professor at the Department of Computer Engineering. His current research interests include System-on-Chip and Network-on-Chip, and VLSI design and test. He has published more than 100 refereed papers in the related areas. Dr. Hessabi has served as the program chair, general chair, and program committee member of various conferences.

Email: hessabi@sharif.edu

Paper Handling Data:

Submitted: 07.04.2019
Received in revised form: 31.07.2019
Accepted: 11.08.2019
Corresponding author: Siavash Bayat Sarmadi
Computer Engineering Department, Sharif University of Technology, Tehran, Iran