RISO: Relaxed Network-on-Chip Isolation for Cloud Processors

Hang Lu[†‡], Guihai Yan[†], Yinhe Han[†‡], Binzhang Fu[†] and Xiaowei Li[†‡]
[†]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[‡]University of Chinese Academy of Sciences, Beijing, China
{luhang, guihai_yan, yinhes, fubinzhang, lxw}@ict.ac.cn

ABSTRACT

Cloud service providers use workload consolidation technique in many-core cloud processors to optimize system utilization and augment performance for ever extending scale-out workloads. Performance isolation usually has to be enforced for the consolidated workloads sharing the same many-core resources. Networks-on-chip (NoC) serves as a major shared resource, also needs to be isolated to avoid violating performance isolation. Prior work uses strict network isolation to fulfill performance isolation. However, strict network isolation either results in low consolidation density, or complex routing mechanisms which indicates prohibitive high hardware cost and large latency. In view of this limitation, we propose a novel NoC isolation strategy for many-core cloud processors, called *relaxed isolation (RISO)*. It permits underutilized links to be shared by multiple applications, at the same time keeps the aggregated traffic in check to enforce performance isolation. The experimental results show that the consolidation density is improved more than 12% in comparison with previous strict isolation scheme, meanwhile reducing network latency by 38.4% on average.

Categories and Subject Descriptors

C.1.2 [**Multiple Data Stream Architectures (Multiprocessors)**]: Interconnection architectures; C.1.4 [**Parallel Architectures**]: Distributed architectures; D.4.1 [**Process Management**]: Concurrency, Threads

General Terms

Performance, Design, Algorithms

Keywords

Networks-on-Chip, Cloud Computing, Cloud Processors, Workload Consolidation, Performance Isolation, Relaxed Isolation

1. INTRODUCTION

Cloud computing has emerged as a fundamental platform to deploy increasing on-line services like web searching, social networks and so forth. To tackle expanded power and space consumption brought by the growing cloud infrastructure, the processor with tens even hundreds of cores is believed to play a critical role in the coming cloud computing era, or simply called many-core "cloud processors". Intel's 48-core "Single-chip Cloud Computer" [1] and Tilera's "TILE-Gx 3000 Series" [2] are two representatives. The cloud processor chips can accommodate abundant parallelism for concurrent **scale-out workloads**, which demand various amounts of computational resources and will be scheduled back and forth by the data center software [3][4]. For such cloud computing platform, one key optimization is to reduce the TCO (total cost of ownership) to avoid low hardware utilization by aggressive "workload consolidation" technique [5][6]. In such computing paradigm, **performance isolation** between scale-out workloads has to be enforced to provide controllable QoS and priority-based services which are critical for cloud computing service providers, such as Google and Amazon.

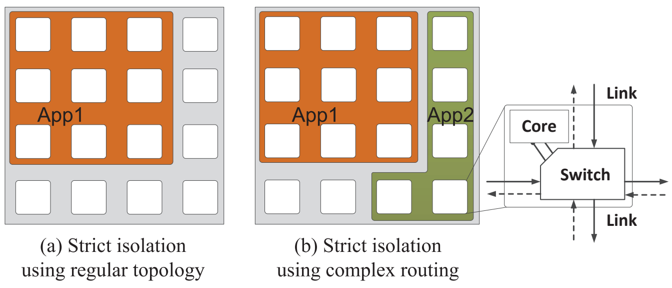

App1
App1
App2
Core
Switch
Link
Link
(a) Strict isolation using regular topology
(b) Strict isolation using complex routing

Figure 1: Two traditional schemes of strict isolation

Performance isolation imposes two orthogonal requirements: 1) Computation isolation, i.e. the computing cores, and the associate cache, memory bandwidth etc., should not be preempted by other workloads in uninterruptable computing sessions [6][7]. 2) Communication isolation, or network isolation, i.e. the on-chip network (NoC) traffic of different workloads should not block each other because the performance of many parallel applications are highly sensitive to network latency [8]. Our work focuses on communication isolation, the same important but less extensively studied topic than computation isolation.

Communication isolation strategies involve making tradeoffs between the regularity of network topology and complexity of routing mechanism. Usually, a regular topology, e.g. rectangular-shaped network, has more efficient routing algorithm, but lower server consolidation density, hence less hardware utilization. By contrast, enabling the network isolation to support more flexible topologies, thereby achieving high consolidation density, will inevitably complicate the routing mechanism. This tradeoff can be further explained with the following example.

Figure 1(a) shows the principle of regular-shaped isolation. The 16-core processor accommodates one application (App1) mapped into a rectangular-shaped region. However, the consolidation density is poor to meet the regular shape requirement: there are seven routers which have to be idle although the incoming application (App2) only needs a five-router network. To relax this limitation, a viable solution is to enable the network isolation to support irregular shapes through implementing flexible routing mechanisms such as Up*/Down* [9] or table-based routing [10], as Figure 1(b) shows. However, the cost of employing those complex routing can hardly be justified for cloud processors given the prohibitive TCO and sporadic performance variations.

Previous work fails to resolve the above contradiction between consolidation density and complexity of routing mechanisms. The reason is that those schemes follow the concept of "**Strict Isolation**", i.e. resorting to strict physical isolation to achieve performance isolation. However, enforcing such strict rule is quite conservative and often leads to overdesign. The on-chip routers and links are often heavily underutilized, especially those on the application region boundaries. By judiciously sharing some routers and links, we can still achieve performance isolation without nailing down to strict isolation. This paper thereby proposes the concept of "**Relaxed Isolation (RISO)**" for many-core cloud processors. Following this concept, we can achieve the consolidation density of schemes supporting irregular topology, but employ the same efficient routing mechanism as in schemes using regular topology. In particular, this paper makes the following contributions:

- *We propose relaxed NoC isolation scheme enforcing workload performance isolation.* We find the traditional strict isolation is highly conservative for performance isolation, which impairs the consolidation density. RISO does not resort to physical isolation, but permits conditional network resource sharing without violating performance isolation.
- *We propose an application mapping algorithm to maximally exploit the potential of RISO.* This algorithm is fully with respect to performance isolation by preventing overlaid traffic exceeding a safe threshold. Also, irregular-shaped regions, which are wasted in previous work to compromise with routing complexity, are also taking into consideration to further improve consolidation density.

The rest of this paper is organized as follows. Section II describes the motivation of RISO by further specifying the limitation of traditional strict isolation schemes. Section III presents the key algorithms to implement RISO. Section IV shows experiment setup and results, followed by related work in Section V and conclusion in Section VI.

2. MOTIVATION

2.1 "Strict Isolation" Degrading Consolidation Density

Improving consolidation density is an effective approach to reduce the TCO of data centers for cloud service providers. In the future, the effectiveness of consolidating multiple workloads into a single many-core cloud processor is just like today's sever consolidation across racks in data centers [5]. However, enforcing performance isolation would be more challenging in cloud processors because the on-chip network is much less flexible than its traditional off-chip counterpart. In particular, the routing mechanism for large-scale many-core system usually follows dimensional order routing (DOR) which is table-less and tailored for on-chip networks [11]. DOR imposes network topology constraints for every workload. By the doctrine of "strict isolation", the consolidation density will be degraded. For example, Figure 2(a) shows dynamic application mapping over time. In this example, App1, App2, $\cdots$, and App5 have already been mapped into the system at that time. The remaining free cores constitute a contiguous but irregular region. Supposing a ten-core workload, App6, is waiting to be served. However, the DOR topology constraint renders this application fail to be mapped because a regular rectangle shape cannot be found under "strict isolation", although there are still 16 free cores which is more than App6 required. Furthermore, the block of App6 probably prevents the subsequent applications, for example App7, from execution, which further degrades the consolidation density.

2.2 Breaking Strict Isolation — RISO

We propose *Relaxed Isolation (RISO)* to tackle the limitation of *strict isolation*. We find that using strict isolation in on-chip networks to ensure communication isolation is highly

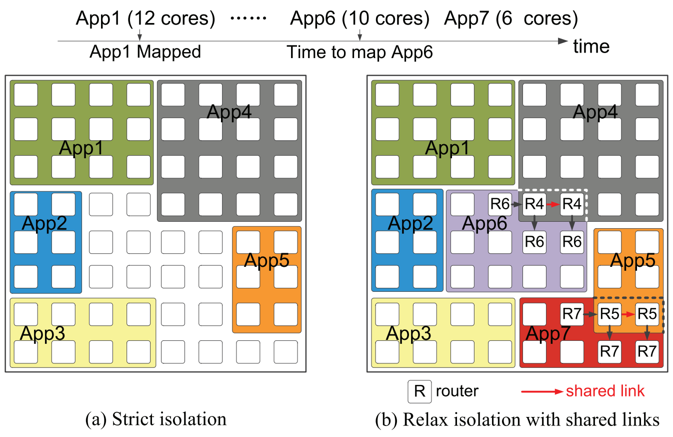


Figure 2: Application mapping on strict isolation and RISO

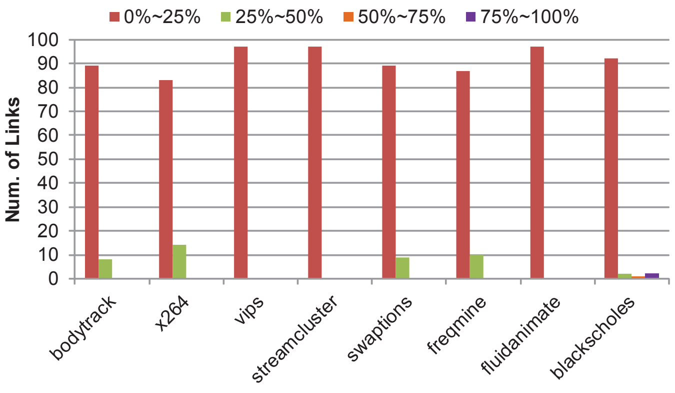


Figure 3: Link utilization distribution

conservative. Since our ultimate goal is performance isolation, the App6 can be mapped into the irregular regions as long as the aggregated traffic on the "overlapped" routers and links won't degrade the latency of each other. As Figure 2(b) shows, by permitting the router and link sharing — RISO, both App6 and subsequent App7 can be served without delay, hence improving the consolidation density.

The rationale behind RISO is to exploit under-utilized routers and links. We find that low NoC utilization is not uncommon in reality, which sufficiently justifies the concept of RISO. The utilization (U) of a link within N link cycles is calculated by Eq.(1) [12].

$$U = \frac{\sum_{i=0}^{N} \alpha(i)}{N} \qquad \alpha(i) = \begin{cases} 0 & \text{Link is } \textit{idle} \text{ at cycle } i; \\ 1 & \text{Otherwise.} \end{cases} \tag{1}$$

We survey the link utilization of a set of workloads. The result is shown in Figure 3, represented by "histgram". The link utilization is divided into four ranks: 0∽25%, 25∽50%, 50∽75%, 75∽100%. The result shows that the lowest rank, 0%∽25%, dominates in all applications. Very few links can reach up to the second rank, without mention the third and forth ranks. Hence, such severely low utilization should provide a unique opportunity for RISO.

RISO won't violate communication isolation as long as the aggregated traffic is kept below a certain threshold, called "congestion point" in this paper, as Figure 4 shows. We find that network latency starts to increase only when the link utilization increases beyond the congestion point. This trend applies to various traffic patterns. Experimental study shows the congestion point is application-independent and at the link utilization around 65%, which agrees with prior work [13]. Given that link utilization is usually much less than 25% in reality, we can safely conclude that the minority of link sharing in RISO won't cause obvious latency increase,

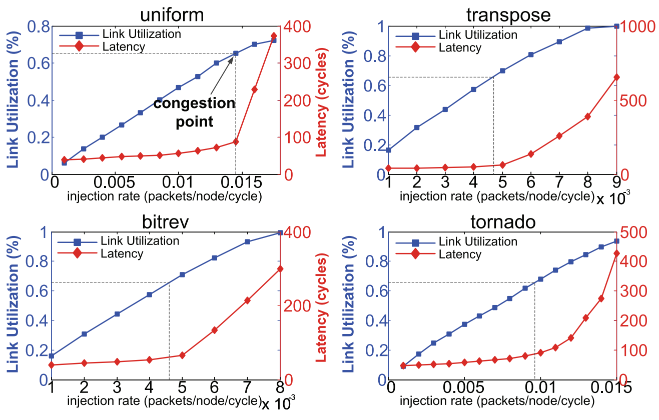


Figure 4: Link utilization and latency under various traffic patterns

therefore keeping the performance isolation intact.

In this paper, we confine our scope to many-core cloud processors connected with "Mesh" networks, which has been proved with superior scalability. But the concept of relaxed isolation is applicable to other types of on-chip networks.

3. APPLICATION MAPPING ALGORITHM BASED ON RISO

Mapping application into a many-core system is to assign the application a specified number of physical cores whose network is organized to the routing-allowed topology, meanwhile without violating the criteria of performance isolation. The basic mapping procedure can be divided into two steps: 1) Search for the physical core groups (regions) which meet the topology requirement under the condition of relaxed NoC isolation, instead of strict isolation; 2) Verify the performance isolation criteria by checking for the utilization of shared links carrying the overlapped traffic. The following will delve into the details of the proposed mapping algorithm.

3.1 Topology Representation

For some applications with intensive intra-application communications, the performance and power consumption can be topology-specific [14][15]. We therefore assume each application to be mapped has a *preferred topology*, which serves as an input to our mapping algorithm. An application's preferred topology incorporates two unique characteristics: *physical shape* and *threads organization*. The proposed algorithm first searches for the candidate regions in NoC that can accommodate the preferred physical shape. To maximize the consolidation density, the mapping algorithm should be capable to handle not only regular shape, i.e. rectangle, but also various irregular shapes ignored in previous works [16]. We abstract those irregular shapes into the following three basic types: "L", "⊢", "⊏", as Figure 5 shows, with various rotations and mirrors.

To represent the physical shapes in Figure 5, we define *horizontal vector*, $H[h_1, h_2, h_3, \cdots ., h_n]$, and *vertical vector*, $V[v_1, v_2, v_3, \cdots, v_n]$, where h_n and v_n is the number of cores in the nth row and column respectively in the preferred shape. H applies to shapes that exhibit complete contiguity in horizontal dimension. For instance, Figure 6(a) shows a "L" shape: the 1st and 2nd row each requires 4 cores; the 3rd and 4th row each requires 2 cores and every row is contiguous; hence this shape is represented by the horizontal vector $H[4, 4, 2, 2]$. However, for the shape that is non-contiguous horizontally but contiguous vertically, we use a vertical vector as Figure. 6(b) shows. Its corresponding vertical vector is $V[4, 4, 2, 2, 2, 4, 4]$. The other irregular shapes both H and V cannot describe are beyond the scope of this paper.

For the multiple threads of an incoming application, we use the tuple $\langle t_i, p_j \rangle$ to indicate each thread and its corresponding position in the physical shape. Specifically, parameter t_i

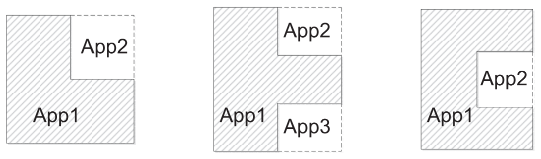
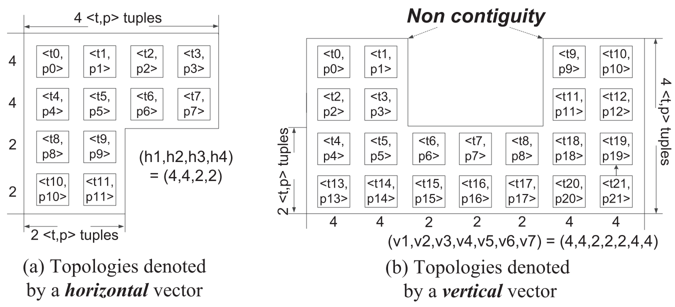


Figure 5: RISO supported shapes



4 <t,p> tuples
4
4
2
2
(h1,h2,h3,h4) = (4,4,2,2)
2 <t,p> tuples
(a) Topologies denoted by a *horizontal* vector
Non contiguity
2 <t,p> tuples
4 <t,p> tuples
4 4 2 2 2 4 4
(v1,v2,v3,v4,v5,v6,v7) = (4,4,2,2,2,4,4)
(b) Topologies denoted by a *vertical* vector

Figure 6: Shapes represented by horizontal (H) or vertical vector (V), and thread organization

and p_j means that the ith thread is located at the jth position, as Figure 6 shows. We use a tuple set $S = \{\langle t_i, p_j \rangle\}$ to represent all threads of an application and their positions in the physical shape.

3.2 Problem Formulation

Based on the above specification, the problem can be formulated as follows:

• Given: 1) the NoC topology, $T(F, B)$, where F and B indicates the set of *free* and *busy* cores, respectively; 2) the traffic matrix, $M_{running}$, whose elements denote the historical communication volume between thread pairs of various applications; 3) the preferred topology, denoted by H or V and tuple set S; 4) the link utilization threshold $U_{congest}$ under which performance isolation can be enforced.

• Determine: 1) the mapping of every $< t_i, p_j >$ tuple from S to F, $< t_i, p_j >$: $S \rightarrow F$. After mapping, relevant position of threads remains the same as in preferred topology; 2) the shared link set L, and link utilization U of every shared link $l \in L$;

• With respect to the constraint: $\forall l \in L$, $U_l < U_{congest}$.

3.3 Algorithm in Detail

Given the inputs and constraints, the mapping algorithm solves the problem in two steps: 1)Topology searching, represented in Algorithm 1; 2) Performance verification, described in Algorithm 2.

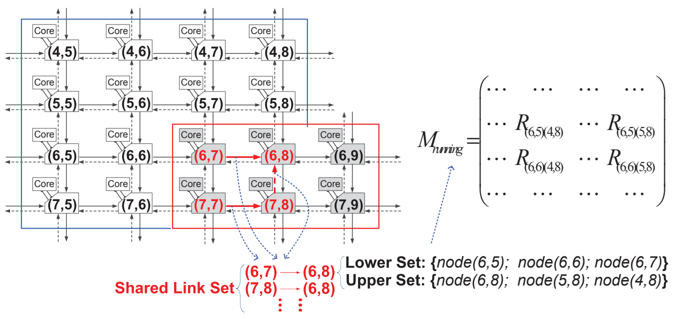
Step 1: Topology Searching: The algorithm will firstly search the NoC for proper candidate shapes to serve the incoming application. If the number of free cores in F is fewer than that the application requires (number of $< t_i, p_j >$ tuples in S), the searching process returns directly with a failure. Otherwise, it tries to find the candidate shapes specified by the H or V. The detail of topology searching is described in Algorithm 1. Line 3 through line 13 are responsible for searching the target shape denoted by H (horizontal vector in this example). The algorithm starts searching T column by column (line 5) to satisfy every element in H. If it finds a busy node (line 8), the algorithm starts searching from another node in F. As long as every element in H is satisfied, the target topology is found (line 4 to 11). Otherwise, if the algorithm has traversed all nodes in set F but still does not find shape identical to H (line 13), the algorithm returns with a failure. Algorithm 1 aims at shapes indicated by the horizontal vector, and for the shapes indicated by vertical vector (V), the overall process is the same, except that it searches T row by row to satisfy every element in V (line 5). Note that Algorithm 1 is not limited to searching 'L' shape, and to further boost consolidation density, it is also applicable to

Algorithm 1 Topology Searching

```
Input: Requested shape: $H$ or $V$; NoC topology: $T(F, B)$;
Output: $Found$ or $NotFound$
 1: if $H$ then
 2:   $h_{len}$ = length($H[h_1, h_2, \ldots, h_i, \ldots, h_n]$);
 3:   for each $Node(row, col) \in F$ do
 4:     for $i = 0; i < h_{len}; i++$ do
 5:       if $\{[Node(row+i, col), Node(row+i, col+h_i)]\} \subseteq F$ then
 6:         continue;
 7:       else
 8:         break; //there is a $Node \in B$, start from another $Node \in F$
 9:       end if
10:     end for
11:     return $Found$; //the shape denoted by $H$ is found in $T$
12:   end for
13:   return $Notfound$; //no shape is identical to $H$ in $T$
14: end if
```

Algorithm 2 Performance Verification

```
Input: the traffic matrix, $M_{running}$, shared link set, $L$; time interval, $time$, link utilization threshold, $U_{congest}$;
Output: The validity, $validity$;
 1: int $sum = 0$;
 2: for each shared link $l(<(x_{from}, y_{from}), (x_{to}, y_{to})>) \in L$ do
 3:   for each $node(a, b) \in lower\_set$ of $l$ do
 4:     for each $node(c, d) \in upper\_set$ of $l$ do
 5:       $sum$ += $R_{(a,b)(c,d)}$; //sum up the values in traffic matrix
 6:     end for
 7:   end for
 8:   $U = \frac{sum}{time}$;
 9:   if $U > U_{congest}$ then
10:     $validity = false$; //congestion will happen after mapping
11:   end if
12: end for
13: $validity = true$; //$U$ of every shared link is lower than $U_{congest}$, performance isolation is ensured
```



Core
(4,5)
(4,6)
(4,7)
(4,8)
(5,5)
(5,6)
(5,7)
(5,8)
(6,5)
(6,6)
(6,7)
(6,8)
(6,9)
(7,5)
(7,6)
(7,7)
(7,8)
(7,9)
$M_{running} = \begin{pmatrix} \cdots & \cdots & \cdots & \cdots \\ \cdots & R_{(6,5)(4,8)} & \cdots & R_{(6,5)(5,8)} \\ \cdots & R_{(6,6)(4,8)} & \cdots & R_{(6,6)(5,8)} \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}$
Shared Link Set
(6,7) → (6,8)
(7,8) → (6,8)
Lower Set: {node(6,5); node(6,6); node(6,7)}
Upper Set: {node(6,8); node(5,8); node(4,8)}

Figure 7: Shared link set identification

other shapes shown in Figure 5.

Step 2: Performance Verification: After successfully finding the target topology, we need to verify the criteria of performance isolation, represented by the constraint that the aggregated link utilization of every shared link U_l will not exceed $U_{congest}$. Firstly, we need to figure out the shared link set L. As an example, Figure 7 shows the scenario after mapping topology in Figure 6(a). Every node has a coordinate marked by the horizontal and vertical location in NoC. A link is denoted by the coordinates of associate nodes as $<(x_{from}, y_{from}), (x_{to}, y_{to})>$. For example in Figure 7, there are three shared links in DOR routing mechanism. The links can be presented as $<(6,7),(6,8)>, <(7,7),(7,8)>$ and $<(7,8),(6,8)>$, respectively.

Secondly, to calculate the aggregated traffic on a shared link, we divide the associate nodes of the shared link into two sets: *lower_set* and *upper_set*, defined by Eq. 2. In DOR, a shared link will only carry the traffic generated from nodes in *lower_set* and terminated at nodes in *upper_set*. By looking up into the traffic matrix $M_{running}$, we can get the aggregated traffic volume on the shared link.

$$\begin{cases} lower\_set : \{Node(x, y) \mid x = x_{from} \&\& y \leq y_{from}\} \\ upper\_set : \{Node(x, y) \mid x \leq x_{to} \&\& y = y_{to}\} \end{cases} \tag{2}$$

Algorithm 2 shows in detail the verification process. Line 3 through line 7 determines the aggregated traffic volume on each shared link. U is calculated based on Eq. 1 in line 8. Line 10 indicates that if the U of any shared link exceeds $U_{congest}$, this candidate topology is not valid and must search another candidate from Step 1. It is valid only if all shared links are satisfied, as line 13 describes.

Note that our algorithm takes the "first-fit" principle [16]. That is if multiple topology candidates can pass the Step 2, we take the first one.

3.4 Traffic Prediction

As many prior NoC flow management solutions, RISO relies on accurate traffic prediction. At each scheduling interval, the operating system, based on the traffic history, predicts the traffic distribution and thereby identifies the sharable links to implement RISO. Most prior work uses liner predictor to fulfill this purpose. We find that although liner predictor is capable for gradually-changed traffic patterns, it's highly unreliable to cope with bursty traffic patterns which, if fail to predict, can jeopardize the performance isolation. Therefore, we take a "conservative" approach in traffic prediction: for bursty traffic, since the traffic volume changes sharply, we just exclude the associated links from sharing. This may slightly degrade the consolidation density, but the performance isolation is well guaranteed. Specifically, we modified last value predictor (LVP) [17] to handle both bursty and non-bursty scenarios. The prediction function is implemented as Eq. 3.

$$T_{prediction} = \begin{cases} h_2 & \mid \frac{(h_2 - h_1)}{h_1} \mid < 10\%; \\ +\infty & Otherwise. \end{cases} \tag{3}$$

The predictor stores two most recent traffic volumes as h_1, h_2 for every source-destination pair. If the two values differ sharply, we exclude the associated links from sharable links by assigning a $+\infty$ to the final prediction value; otherwise we still follow the LVP. We find that setting the bar to 10% works well.

4. EVALUATION

4.1 Experimental Setup

4.1.1 *Metric for Consolidation Density*

We use *system utilization* (U_{system}) [16] as a metric for consolidation density evaluation. For a N-node system during T period of time, U_{system} is defined by Eq. 4.

$$U_{system} = \frac{\sum_{i=1}^{N} T_i}{N \times T} \tag{4}$$

where T_i is the busy time of node i over T period of time. A high system utilization means high consolidation density.

System utilization depends on the "*load*" condition [16] which is defined by Eq. 5.

$$Load = \frac{R \times S}{N \times I} \tag{5}$$

where R is average requested resources (i.e. cores in this paper) of all applications; I is average inter-arrival time between consecutive applications; S is average application running time. *Load* below 1 means application arrival rate is lower than departure rate; otherwise the system will be overloaded and improving system utilization will be critical. The values of these parameters used in the experiment are listed in Table 1.

4.1.2 *Performance Simulation Setup*

We modified Booksim2.0 [18] to evaluate performance after using the proposed application mapping algorithm. The

Table 1: Parameters of system utilization evaluation

Parameter	Value
Topology	Mesh (32*32 and 16*16)
Scheduling mechanism	FCFS
R	64
S	2000 (in cycles)
distribution of requested Num. of cores	uniform
Num. of consolidated workloads	10000 (per experiment)
load range	[0.1∽1.6], step by 0.1

baseline NoC topology is a 8x8 mesh. The router is configured with a two-stage pipeline plus one cycle for link traversal. We use two virtual channels and each has an eight flit buffer. The congestion threshold ($U_{congest}$) is set at 65%, in accordance with the result shown in Figure 4.

4.1.3 Workloads

For consolidation density evaluation, we map 10000 consolidated workloads at each *load* condition. We log the U_{system} in real time. This measurement is repeated ten times at each *load*. The final U_{system} result is the average of the ten measurements.

For performance evaluation, we run application traces [8] and each application acts as a workload. Application traces are obtained from GEMS [19], a full system simulator. The detailed configuration is shown in Table 2.

Table 2: Full system simulator configuration

Parameter	Value
Cloud Processor	16 in-order cores
Coherence Protocol	MOESI
L1 I/D Cache	32KB (2-way associative)
L1 Cache Access Latency	1 cycle
Private L2 Cache	256KB (4-way associative)
L2 Cache Access Latency	8 cycles
Main Memory Latency	90 cycles

4.1.4 Baselines Compared

We compare our scheme with two previous representative schemes: the first scheme employs efficient routing but at the expense of lower consolidation density [20], denoted by "Regularity-oriented" scheme. Clearly, "Regularity-oriented" scheme ideally enforces performance isolation. The second scheme takes the opposite, i.e. emphasizing the density, but paying for more complex routing mechanisms [16], denoted by "Density-oriented" scheme.

In the experiment we will show that our scheme, "RISO", can also preserve performance isolation as "Regularity-oriented" scheme provides, and meanwhile achieves the consolidation density of "Density-oriented" scheme, but with much better performance.

4.2 Result 1: Consolidation Density

Figure 8 shows the system utilization from underload to overload scenarios. The result shows that *RISO* improves the system utilization by up to 12% higher than *Regularity-oriented* scheme in the overload condition. Surprisingly, RISO performs almost equally well to *Density-oriented* scheme (within 0.1%). Even though RISO cannot exploit all irregular regions due to the link utilization constraint, but it can deal with some unique regions such as "⊏" which cannot be supported in "Density-oriented" scheme, which makes our scheme match density-oriented scheme in consolidation density.

Moreover, our scheme uses DOR as the underlying routing algorithm which is more efficient and cost-effective regarding network performance, as the following results show.

4.3 Result 2: Performance

4.3.1 Performance Isolation Analysis

Regularity-oriented schemes are regarded for ideal performance isolation by enforcing strict physical isolation, as described in Section II. Hence, we compared our relaxed isolation scheme to the regularity-oriented scheme to verify the capability of performance isolation. We run application traces

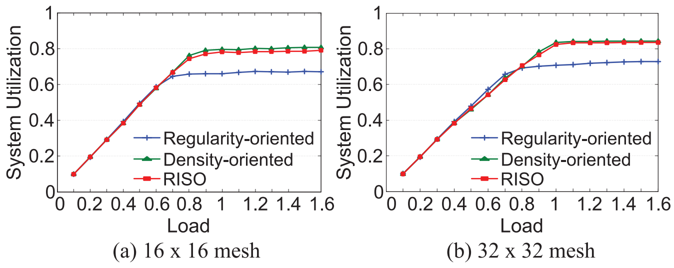


Figure 8: System utilization for 16x16 and 32x32 mesh

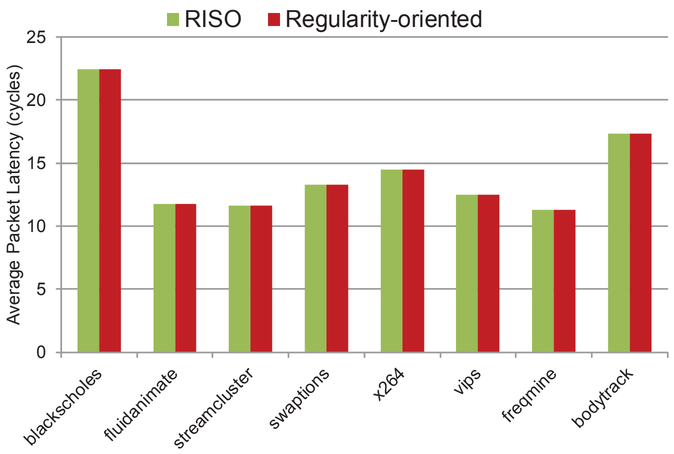


Figure 9: Performance comparison between routing-oriented scheme and RISO

under RISO and regularity-oriented condition respectively. The results are shown in Figure 9. RISO won't degrade the latency compared to the regularity-oriented scheme. This is because we put a hard constraint for each shared link, as Section III describes. Therefore, we can safely conclude that RISO preserves performance isolation, but with higher consolidation density as shown in Figure 8.

4.3.2 Network Latency Analysis

Density-oriented scheme is notorious for traffic latency, although it can enforce performance isolation and provide high consolidation density. Hence, we show how much latency can be improved with RISO. We run various combinations of two application traces under relaxed and density-oriented isolation respectively. Note that RISO uses efficient DOR, while density-oriented scheme uses the most favorable Up*/Down* [9] routing mechanism.

Figure 10 shows the comparison results for 25 groups of consolidated applications. Clearly, our scheme wins for all. The latency, if using density-oriented scheme, will degrade over 100% for some groups such as `blackscholes_bodytrack` (114%) and `swaptions_bodytrack` (101%), which are usually the mix of computation intensive and memory intensive applications. Such combination usually renders more under-utilized links which RISO can take advantage of while density-oriented scheme cannot. The average latency improvement is 38.4% for 25 groups of consolidated applications.

5. RELATED WORK

The concept of RISO is similar to the topology virtualization techniques proposed in [21] and [22], in which NoC is reconfigured and the software always observes a logically regular-shaped topology when faults are happening. However, our work targets the mapping of the arbitrary-shaped topology required by an application into the appropriate region to improve consolidation density.

Quality of service (QoS) in NoC level is introduced to fairly allocate on-chip resources according to specific service poli-

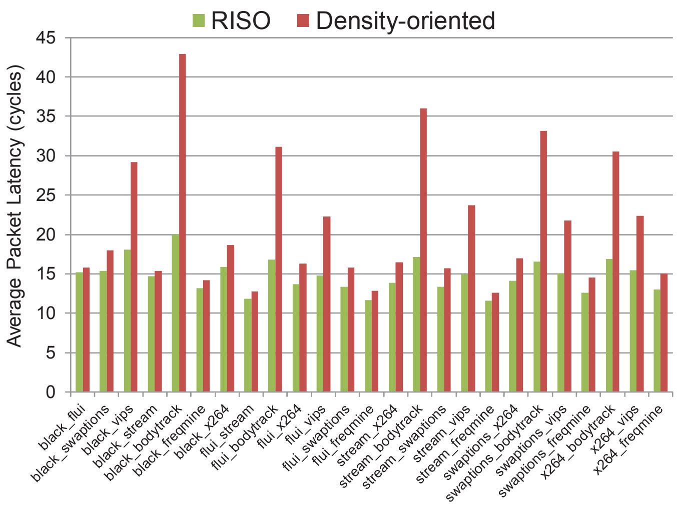


Figure 10: Latency comparison between density-oriented scheme and RISO

cies. For instance, Brand et al [13] uses link utilization as the congestion measure, and proposed that the bandwidth and latency will be guaranteed for best-effort communication service if the contribution to link utilization by different workloads stays below a certain threshold. In [23], virtual point-to-point links are used to ensure reliable communication if the threads of an application are distributed into disjoint NoC regions. Our work is orthogonal to them.

Performance isolation technique is highly required to achieve controllable QoS in NoC level, and is firstly introduced in [24]. It clarifies the basic items needed to solve in this area. Particularly, the tradeoff between regularity of topology and complexity of routing is the most important in that it relates directly to the network performance and power consumption.

Some proposed techniques follow the strict NoC isolation strategy using rectangular shapes for performance isolation, such as [20]. These methods are restrained by the maximum number of consolidated workloads, which will degrade the consolidation density. Unlike those regular shaped performance isolation methods, Solheim et al. [16] proposed an irregular-shaped isolation based on complex routing mechanism. This method also follows strict isolation between workloads and to some extent improves consolidation density compared with rectangle based isolation. However, its routing mechanism is less efficient and exhibits substantial degradation with respect to network performance. To the best of our knowledge, the proposed relaxed isolation scheme is the first work that simultaneously takes the advantage of regularity of topology and efficient routing mechanisms in network isolation.

6. CONCLUSION

This paper proposes relaxed isolation (RISO) strategy to enforce performance isolation constraint in workload consolidation. Unlike the traditional strict isolation strategy such as regularity-oriented and density-oriented approach, RISO allows under utilized links to be shared by multiple applications, as long as the aggregated contribution to link utilization is lower than the congestion threshold. Compared with regularity-oriented approach, RISO supports more flexible topologies and can greatly improve consolidation density. RISO does not complicate the routing mechanism required by density-oriented approach; it also uses efficient and cost-effective DOR routing mechanism and hence yields higher network performance. In other words, RISO strikes better tradeoff between regularity of topology and complexity of routing algorithm in implementing network isolation. We therefore believe that RISO is a promising scheme for workload consolidation in many-core cloud processors.

7. ACKNOWLEDGEMENT

This work is supported in part by National Basic Research Program of China (973) under grant No. 2011CB302503, in part by NSFC under grant No. (61202056, 61100016, 61076037, 60921002). Correspondence should be addressed to Prof. Yinhe Han. We also thank Prof. Ninghui Sun and Prof. Lixin Zhang for their support.

8. REFERENCES

[1] Intel Single-chip Cloud Computer, http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-computer.html.

[2] TILE-Gx 3000 Series Cloud Processor, http://www.tilera.com/products/processors/TILE-Gx-3000.

[3] M. Ferdman et al, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *ASPLOS 2012*, pp. 37–48.

[4] P. Lotfi-Kamran et al, "Scale-out processors," in *ISCA 2012*, pp. 500–511.

[5] Amazon Elastic Cloud Computing, http://aws.amazon.com/ec2/.

[6] M. Marty and M. Hill, "Virtual hierarchies to support server consolidation," in *ISCA 2007*, pp. 46–56.

[7] S. Ma, N. Jerger, and Z. Wang, "Dbar: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *ISCA 2011*, pp. 413–424.

[8] C. Bienia et al, "The parsec benchmark suite: characterization and architectural implications," in *PACT 2008*, pp. 72–81.

[9] M. Schroeder, "Autonet: A high-speed, self-configuring local area network using point-to-point links," tech. rep., 1990.

[10] A. Mejia et al, "Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori," in *IPDPS 2006*, pp. 10–19.

[11] S. Bell et al, "Tile64 - processor: A 64-core soc with mesh interconnect," in *ISSCC 2008*, pp. 88–99.

[12] S. Li, L. Peh, and N. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *HPCA 2003*, pp. 91–102.

[13] van den Brand et al, "Congestion-controlled best-effort communication for networks-on-chip," in *DATE 2007*, pp. 1–6.

[14] M. Kandemir, O. Ozturk, and S. Muralidhara, "Dynamic thread and data mapping for noc based cmps," in *DAC 2009*, pp. 852–857.

[15] B. Fu, Y. Han, and J. Ma, "An abacus turn model for time/space-efficient reconfigurable routing," in *ISCA 2011*, pp. 259–270.

[16] A. Solheim et al, "Routing-contained virtualization based on up*/down* forwarding," in *HiPC 2007*, pp. 500–513.

[17] Y. Huang et al, "Ntpt: On the end-to-end traffic prediction in the on-chip networks," in *DAC 2010*, pp. 449–452.

[18] Booksim2.0, https://nocs.stanford.edu/.

[19] GEMS, http://research.cs.wisc.edu/gems/publications.html.

[20] V. Gupta and A. Jayendran, "A flexible processor allocation strategy for mesh connected parallel systems," in *ICPP 1996*, pp. 166–173.

[21] L. Zhang and Y. Han et al, "Defect tolerance in homogeneous manycore processors using core-level redundancy with unified topology," in *DATE 2008*, pp. 891–896.

[22] L. Zhang and Y. Han et al, "On topology reconfiguration for defect-tolerant noc-based homogeneous manycore systems," in *TVLSI*, pp. 1173–1186, 2009.

[23] M. Asadinia et al, "Supporting non-contiguous processor allocation in mesh-based cmps using virtual point-to-point links," in *DATE 2011*, pp. 1–6.

[24] J. Flich et al, "On the potential of noc virtualization for multicore chips," in *CISIS 2008*, pp. 801–807.