RISO: Enforce Noninterfered Performance With Relaxed Network-on-Chip Isolation in Many-Core Cloud Processors

Hang Lu, Binzhang Fu, *Member, IEEE*, Ying Wang, *Student Member, IEEE*, Yinhe Han, *Member, IEEE*, Guihai Yan, *Member, IEEE*, and Xiaowei Li, *Senior Member, IEEE*

***Abstract*— Workload consolidation is widely used in modern cloud processors to reduce total cost of ownership. Performance isolation has to be enforced between consolidated workloads to achieve controllable quality of service. Networks-on-chip (NoCs), as a major shared resource, often incur traffic interference and violate performance isolation criteria. Previous work resorts to strict isolation strategy that partitions NoC into independent regions to isolate core-to-core communication traffic. However, strict isolation either results in low consolidation density or degrades network performance, and more importantly, cannot be applied to memory access traffic. To address these weaknesses, we propose a novel performance isolation strategy in NoC, called relaxed isolation (RISO). It permits underutilized routers and links to be shared by multiple applications, and, at the same time, it keeps the aggregated traffic in check to enforce performance isolation. Experimental results show that RISO could effectively improve consolidation density and network performance in synergy.**

***Index Terms*— Cloud processor, networks-on-chip (NoCs), performance isolation, relaxed isolation (RISO), workload consolidation.**

I. INTRODUCTION

MORE than 60% cloud computing services are handled in data centers. Total cost of ownership (TCO) is believed to be a major limitation for cloud service providers to deploy scalable online services [1], i.e., web searching and social networks. In a TCO-limited data center, performance per TCO dollar can be boosted by building more efficient hardware architectures to resolve request-level parallelism [2]. Processor with tens even hundreds of cores is hence 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 (SCC) Computer [3] and AMDs Opteron 6000+ Series [4] are two representatives.

For such cloud computing platform, one key optimization to reduce TCO is to avoid low hardware utilization by aggressive workload consolidation technique [5], [6]. Multiple server applications are deployed onto respective virtual machines, which then run simultaneously on the same many-core cloud processor. Many workload consolidation techniques have been proposed in order to increase the execution efficiency, i.e., domain partitioning and dynamic resource reassignment [6]. More importantly, performance isolation must be enforced as well provide controllable quality of service (QoS) and priority-based services.

From workload consolidation point of view, concurrent workloads will most likely interfere with each other in various shared resources, i.e., networks-on-chip (NoCs), and violate performance isolation constraint. When different workloads are injecting traffic for cache coherence or memory access, multiple flows are very likely to collide in the same on-chip router and associate links. Therefore, it is necessary that the traffic of different workloads should be safely isolated from each other, to avoid the performance impact to the latency-sensitive applications [7], [8].

Such communication isolation involves two types of on-chip traffic: 1) core-to-core, including cache coherence [9] or intercore operand transmission [10] and 2) memory access, used to fetch instructions and data from DRAM. Core-to-core isolation incorporates making tradeoffs between the regularity of network topology and complexity of routing mechanism. Usually, a regular topology, i.e., rectangular-shaped network, has more efficient routing algorithm, but lower consolidation density, hence less hardware utilization. In contrast, enabling the communication 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. Fig. 1(a) shows the workload consolidation process over time. The cloud processor is a TILE64 [11] alike many-core with four memory controllers (MCs) at each corner of a 8×8 mesh. In this example, App1–App5 have already been mapped at their time stamp. The remaining free cores constitute a contiguous but irregular region. Suppose a 10-core workload, App6, is waiting to be served. However, the

Manuscript received June 6, 2014; revised October 12, 2014; accepted December 15, 2014. Date of publication January 21, 2015; date of current version November 20, 2015. This work was supported in part by the National Basic Research Program (973) of China under Grant 2011CB302503 and in part by the National Natural Science Foundation of China (NSFC) under Grant 61100016, Grant 61202056, Grant 61221062, and Grant 61376043.

H. Lu is with the University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: luhang@ict.ac.cn).

B. Fu, Y. Wang, Y. Han, G. Yan, and X. Li are with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: fubinzhang@ict.ac.cn; wangying2009@ict.ac.cn; yinhes@ict.ac.cn; yan_guihai@ict.ac.cn; lxw@ict.ac.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2014.2387351

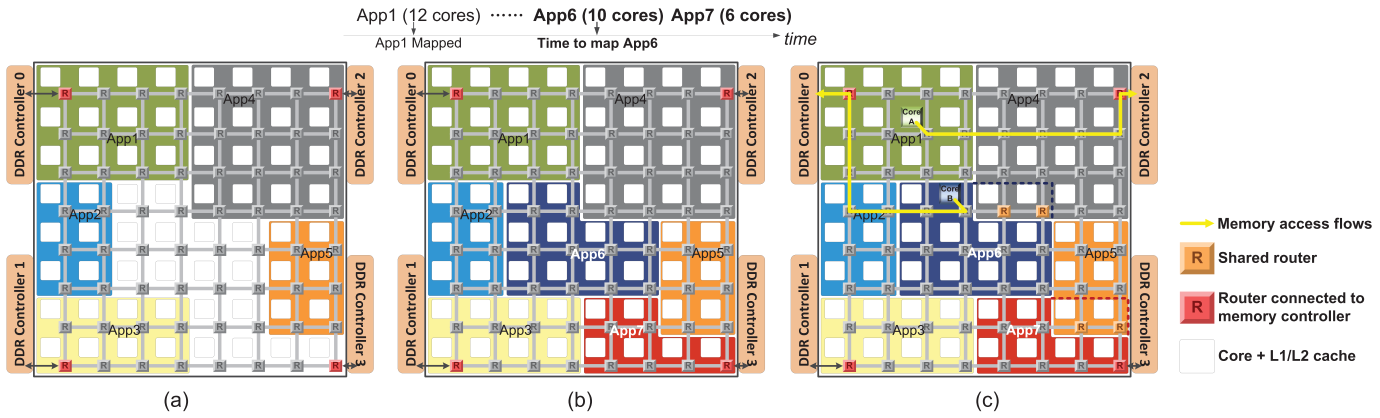


Fig. 1. Two traditional schemes following strict isolation and the proposed RISO. (a) Regularity oriented. (b) Density oriented. (c) RISO.

dimensional order routing (DOR) topology constraint renders this application fail to be mapped because a regular rectangle shape cannot be found, although there are still 16 free cores which is more than App6 required. Furthermore, the blocking of App6 probably prevents the subsequent applications, for example, App7, from execution, which further degrades the consolidation density. To resolve this limitation, a viable solution is to enable the communication isolation to support irregular shapes by implementing flexible routing mechanisms such as Up*/Down* [12] or table-based routing [13], as shown in Fig. 1(b). App6 and App7 could then be consolidated into two irregular-shaped regions. 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 regularity-complexity tradeoff for core-to-core traffic isolation. The reason is that those schemes follow the concept of strict isolation, i.e., resorting to strict region isolation to enforce performance isolation. This philosophy, though straightforward, is quite conservative and often leads to over-design. The on-chip routers and links are often heavily underutilized, especially those on the application region boundaries.

To make things worse, enforcing such strict rule still cannot guarantee the memory access isolation. For commercial many-cores, such as TILE64 [11] or Intel SCC [3], MCs usually locate at fixed chip positions, as shown in Fig. 1. A memory access request must traverse unexpected hops from the source node, i.e., an L2 cache bumping into a READ/WRITE miss, to the destination, i.e., one of the MCs in the corner. The traffic generated from central area of the mesh, i.e., App2 or App6 in Fig. 1(b), will inevitably intrude on other workloads' territory, in order to reach the desired MC, and performance isolation is very likely to be violated if such flows are taken carelessly.

Prior work following strict isolation assumes that MCs are distributed across each tile of the cloud processor [14], [15]. Memory access latency is hence regarded as constant, scratching out the fraction from local last level cache to the desired MC in the corner, which is unpractical in real-world cloud processors.

To address these issues, this paper proposes relaxed NoC isolation (RISO) in workload consolidation. By judiciously sharing some routers and links for different workloads, we can still fulfill performance isolation without nailing down to strict isolation. In particular, this paper makes the following contributions.

1) We propose RISO in NoC to enforce workload performance isolation, which involves two types of on-chip traffic.
   a) *Core-to-Core:* We find that traditional strict isolation is conservative when enforcing performance isolation, which either impairs the consolidation density or complicates the routing. RISO tackles this tradeoff by sharing network resources conditionally, so cost-effective routing could be used without degrading consolidation density.
   b) *Memory Access:* Memory access traffic, for which strict isolation fails to apply, can also fit in RISO framework. A workload is allowed to be mapped closest to the MC it visits most, based on its historical memory access distribution. Such near-MC mapping effectively reduces the end-to-end memory access latency and hence improves the overall network performance.
2) We propose an application mapping algorithm to exploit the maximum potential of the RISO. This algorithm fulfills performance isolation by preventing overlaid traffic exceeding a safety threshold. In addition, irregular-shaped regions, which are wasted in previous work to compromise with routing complexity, are also taken into consideration to further improve consolidation density.

The rest of this paper is organized as follows. Section II describes the motivation of the RISO by further elaborating the limitation of traditional strict isolation schemes. Section III presents the key algorithms to implement RISO. Section IV specifies the experimental platform, metrics, and baselines used for evaluation. Section V shows the results and analysis. Section VI presents the related work. Finally, the conclusion is drawn in Section VII.

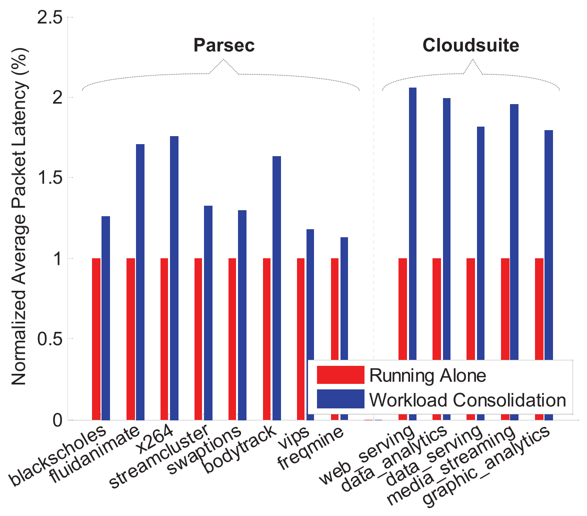


Fig. 2. Interworkload interference, depicted by the height difference of two latency bars.

II. Motivation

A. Interworkload Interference

For consolidated workloads, interference will happen in NoC if network traffic is not regulated properly. This interference will degrade the performance of the executing workloads. For example, Fig. 2 shows the network latencies of eight Parsec [16] and five Cloudsuite [1] benchmark programs, each of which requires eight cores (detailed evaluation framework is shown in Section IV). We study two scenarios: 1) each workload runs alone on a 8×8 mesh-connected NoC and 2) under workload consolidation that a set of workloads are randomly mapped to 64 free cores. For the second scenario, we only calculate the latency of the target workload. Two bars in the figure represent latency results and are normalized to the first scenario. We observe that the interference, which can be illustrated by the height difference of two bars, could be as high as 41% on average for Parsec and even 92.3% for Cloudsuite benchmarks. Even for some computation intensive workloads, i.e., `blackscholes`, the interference would exceed 20%. Such severe interference in server systems may yield consequences in terms of TCO for cloud service providers and QoS commitment for cloud service users.

B. Tackling Regularity/Complexity Tradeoff—RISO

Traditional isolation strategies resort to strict isolation for all consolidated workloads as introduced in Fig. 1(a) and (b). As an alternative, we propose the RISO to address the tradeoff between the region regularity and the routing complexity. 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 would not degrade the latency of each other. As shown in Fig. 1(c), by permitting the router and link sharing—RISO, both App6 and subsequent App7 can be served without delay. This operation has two benefits in terms of communication isolation: 1) improving the consolidation density, due to the employment of irregular shapes and 2) DOR can be applied. For the irregular region that the App6 has been mapped in, the routers and the links shared from App4 serve as a patch to transform the previous

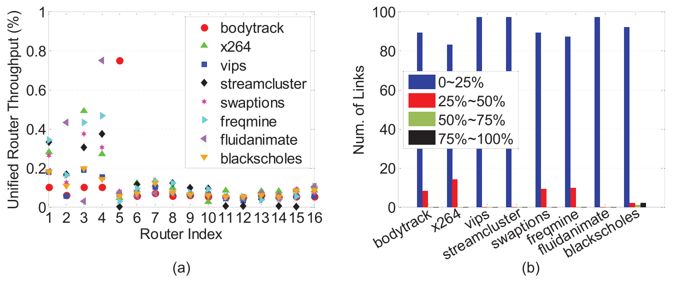


Fig. 3. Two proofs of the feasibility of RISO. (a) Router throughput. (b) Link utilization ranking

irregular shape into a rectangular mesh. DOR, thereby, could be used in the newly generated mesh by the support of the shared region.

The rationale behind the RISO is to exploit underutilized routers and links. Through exploration, we find that low resource utilization is very common in reality, which further justifies the concept of RISO. For example, Fig. 3(a) shows the unified throughput of all routers in the network. It can be seen that nearly 70% routers remain underutilized (throughput never exceeds 15%). Only a small fraction could reach 30% throughput. It provides a unique opportunity that sharing these underutilized routers, rather than monopolizing them may not increase the contention delay of relevant workload packets.

As another evidence, we evaluate link bandwidth usage under the same experimental platform and the result is shown in Fig. 3(b). We use link utilization as the representative [17] shown by histogram. As can be seen in the figure, link utilization is divided into four ranks: 1) 0%–25%; 2) 25%–50%; 3) 50%–75%; and 4) 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 mentioning the third and fourth ranks. Therefore, by moderately sharing the abundant link bandwidth to multiple workloads, relevant packet latency will not be degraded either.

To formalize the metrics of such resource sharing, we need to evaluate when congestion will happen in NoC, hence violating communication isolation of the sharers. Since runtime link utilization can serve as the representative of bandwidth usage, we thus explore its relationship with congestion, as shown in Fig. 4. We use average packet latency as the representative of congestion. Clearly, it starts to increase steeply only when the link utilization increases beyond a certain threshold, namely, congestion point in the figure. It gives the maximum bandwidth that mixed workload traffic can occupy without causing congestion. Experimental study also shows that the congestion point is application independent and at the link utilization around 65%–70% for the employed NoC configuration, which agrees with [18]. Given that the link utilization is usually much less than 25% in reality, we can safely conclude that the minority of router and link sharing in RISO would not cause obvious latency increase, therefore keeping the performance isolation intact. We thus use the congestion point as the upper limit of such resource sharing.

C. Memory Access Isolation

Memory access interference is another main factor that may cause workload performance loss. Long end-to-end access

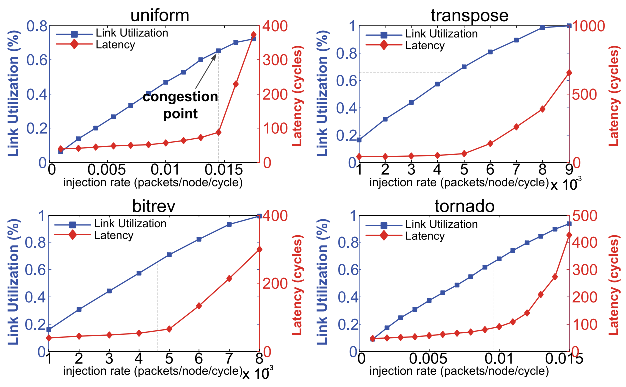


Fig. 4. Congestion point under various traffic patterns, indicating the upper limit of resource sharing in RISO.

latency impacts core execution efficiency. It must wait until the requested instructions and data are fetched from DRAM. Intuitively, memory access traffic is more vulnerable to be blocked because it must cross over other workloads' region to reach the desired MC. This unpredictable traveling results in the uncertainty of memory access latency and may violate performance isolation severely. For example, Fig. 1(c) shows that two traffic flows (marked in yellow), issued by Core A and Core B, respectively, are targeting different MCs.[1] Under DOR, Core A will follow the path Core $A \rightarrow$ App2 $\rightarrow$ App1 $\rightarrow$ MC0. If App2 happens to impose a high network demand, the memory access request will be inevitably blocked in App2. Same situation also happens for the other flow in App4.

To avoid such interference, we cannot resort to strict isolation any more because the memory traffic cannot always be forced to stay within a specific workload region compared with core-to-core traffic. It must be allowed to share data paths with other workload traffic, which makes performance isolation difficult to be enforced. Fortunately, RISO just employs conditional resource sharing, which provides two opportunities for memory access isolation: 1) it enables memory traffic to use shared routers and links heading to the desired MC, while in harmony with other workload traffic and 2) by mapping a workload close to its favorable MC, the possibilities of interference are further reduced. For example, in Fig. 5(a), we can trace the memory access of App6 and identify its most visited MC, i.e., MC0. Based on this historical memory access information, App6 could be mapped to a new region, adjacent to MC0, as shown in Fig. 5(b). This near-MC mapping makes most of the memory requests no longer need to traverse other workload regions, and hence avoids memory access interference with other consolidated workloads. Besides, it shortens the physical distance to reach MC0 for App6 so the memory access latency is also diminished.

In Section III, we will elaborate how the core-to-core and memory access isolations are achieved by deploying

[1]We assume that the continuous data blocks of different workloads are interleaved across multiple memory channels, so that a core is poised to initiate the memory access traffic to any MC. Whereas, other studies [19] that assume channel partitioning" instead of interleaving-based data mapping also fit in RISO framework, because the phenomenon of differential MC affinity is even more evident for mixed workloads.

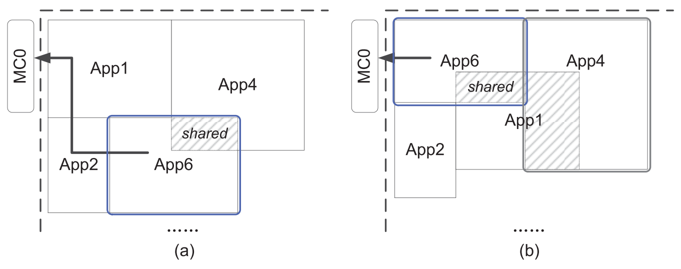


Fig. 5. Near-MC mapping of App6. (a) Initial mapping. (b) Remapped close to the most visited MC0.

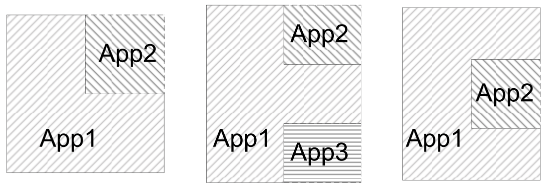


Fig. 6. Preferred topologies supported by RISO.

dedicated application mapping algorithms based on the concept of RISO.

III. Application Mapping Algorithm Supporting RISO

As we know, application mapping methodologies could be classified as design-time and run-time [20] for static and dynamic workload scenarios, respectively. RISO assumes that the target many-core cloud processor employs run-time mapping, and workloads can be scheduled after a fixed execution interval. Each workload hence has the opportunity to be remapped to a new region according to its runtime characteristics and corresponding communication isolation criteria. In this section, we firstly specify that application mapping is actually an optimization problem and then show the details of the proposed mapping algorithm.

A. Problem Formulation

The mapping process involves allocating a specified number of physical cores whose network is organized to the routing-allowed topology. For some applications with intensive intra-application communications, the performance and power consumption can be topology specific [21], [22]. 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: 1) physical shape and 2) threads organization. The proposed algorithm firstly searches for the candidate regions in NoC that is identical to the preferred physical shape. Then, 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 [15]. We abstract those irregular shapes into the following three basic types: 1) L; 2) $\vdash$; and 3) $\sqsubset$, as shown in Fig. 6, with various rotations and mirrors.

To describe the preferred topologies in Fig. 6, we firstly define parameter: 1) horizontal vector: $H[h_1, h_2, h_3, \ldots, h_n]$

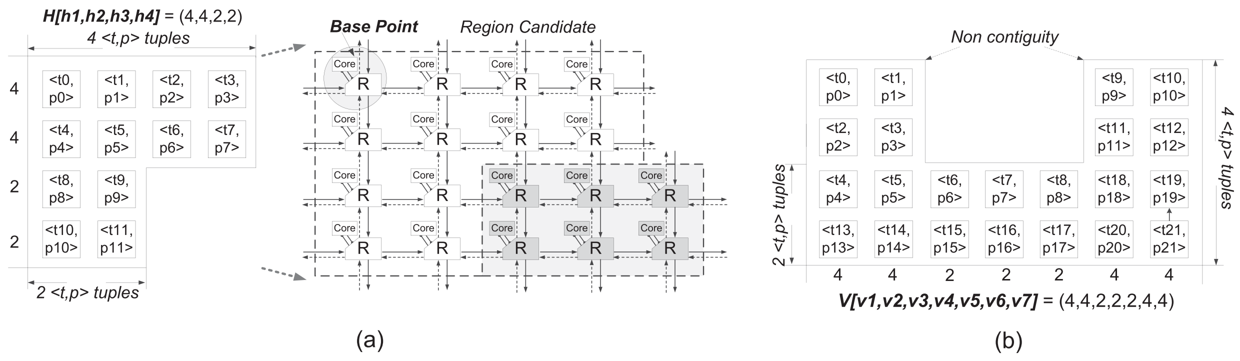


Fig. 7. $< t, p >$ tuples are organized in preferred topologies, which could be represented by the horizontal (H) or vertical vector (V). (a) Topologies denoted by a horizontal vector. (b) Topologies denoted by a vertical vector.

and 2) vertical vector: $V[v_1, v_2, v_3, \ldots, v_n]$, where h_n and v_n are the number of cores in the nth row and column, respectively, in the preferred topology. H applies to shapes that exhibit complete contiguity in horizontal dimension. For instance, Fig. 7(a) shows an L shape: the first and second row each requires four cores; the third and fourth row each requires two 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 noncontiguous horizontally but contiguous vertically, we use a vertical vector as shown in Fig. 7(b). 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. In addition, to fix the preferred topology in the mesh, we use a second parameter base point $\text{BP}(x, y)$ defined as the top-left corner of the preferred topology, as shown in Fig. 7(a). Its coordinates pinpoint the preferred topology on x-/y-axis in a mesh-connected many-core after mapping.

For the forthcoming application threads, we use the $\langle t_i, p_j\rangle$ tuple to represent each thread and its corresponding position in the preferred topology. In particular, parameters t_i and p_j mean that the ith thread resides in the jth position, as shown in Fig. 7. We use set $S = \{\langle t_i, p_j\rangle\}$ to contain all thread-position tuples in the preferred topology.

Based on the above parameter definitions, the application mapping problem can be formulated as follows:

1) *Given*:
   a) NoC topology $T(\text{dim}_x, \text{dim}_y)$, which indicates that the target NoC is a $\text{dim}_x \times \text{dim}_y$ mesh;
   b) Node sets $N(F, B)$, where F and B indicates the set of free and busy cores, respectively;
   c) Traffic matrix M_{running}, which stores historical data used for the prediction of communication volume in the next interval;
   d) MC vector $\text{MC} = \langle \text{mc}_1, \text{mc}_2, \text{mc}_3, \text{mc}_4\rangle$. We assume four MCs, $\langle \text{mc}_i\rangle$ is the memory requests received by controller i;
   e) Preferred topology H or V, and tuple set S;
   f) Link utilization threshold U_{congest}, under which performance isolation can be enforced.
2) *Determine*:
   a) Base point coordinates $\text{BP}(x, y)$;
   b) The mapping of every $\langle t_i, p_j\rangle$ tuple from S to F, $\langle t_i, p_j\rangle$: $S \to F$. After mapping, relevant position of threads remains the same as in preferred topology;
   c) The shared link set L and link utilization U of every shared link $l \in L$.
3) *Minimize*:
   a) $\max\ \text{BP}.y + \begin{cases} \text{length}(H) & \text{horizontal vector} \\ \max(V) & \text{vertical vector} \end{cases}$
   b) $\sum_{(i,j)\in S} D(< t_i, p_j >, \text{MC}_k)$, where $\text{MC}_k = \max(\text{MC})$, D is the *Manhattan distance* from each $< t_i, p_j >$ to MC_k.

[2-DSP]: Application mapping is analogous to packing series of irregular-shaped objects (2-D) into a container with fixed length on x-/y-axis and pack as many objects as possible. We thereby abstract it as a 2-D strip packing (2-DSP) problem. The optimization goal of 2-DSP is to minimize the length on y-axis after packing. Similarly, in RISO, we aim at two goals. First, we need to minimize the maximum length of busy cores on y-axis. $\text{BP}.y$ is the y-coordinate in the mesh after a preferred topology is mapped. If we use horizontal vector as the preferred topology representative, $\text{BP}.y + \text{length}(H)$ hence denotes the maximum length on y-axis. The same concept also applies to vertical vector, except that $\max(V)$ is used to calculate y length. Second, we need to map this workload close to the MC it visits most. Manhattan distance is then used to indicate the aggregate distances from each $\langle t_i, p_j\rangle$ to MC_k.

For the constraints, 2-DSP does not require a fixed length on y-axis, whereas RISO is more conservative because a mesh-connected many-core has fixed length on both dimensions. Therefore, the constraints of our application mapping algorithm are formulated as follows.

1) $\text{BP}\cdot x + \begin{cases} \max(H) & \text{horizontal vector} \\ \text{length}(V) & \text{vertical vector} \end{cases} \in (0, T.\text{dim}_x]$
2) $\text{BP}\cdot y + \begin{cases} \text{length}(H) & \text{horizontal vector} \\ \max(V) & \text{vertical vector} \end{cases} \in (0, T.\text{dim}_y]$
3) $\forall l \in L,\ U_l < U_{\text{congest}}$.

B. Proposed Application Mapping Algorithm

2-DSP is one of the combinatorial optimization problems. We thus do not intend to find its exact polynomial-time solution. Hence, efficient heuristics are introduced in this section.

Algorithm 1 Topology Searching

```
Input: Requested shape: $H$; Node sets: $N(F, B)$;
Output: $candidates$ 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:         $BP(row, col) = node(row, col)$;
12:         $candidates$.push_back(make_pair($H$, $BP(row, col)$));
13:         print "$Found$"; //shape denoted by $H$ is found in $T$
14:     end for
15:     return $Notfound$; //no shape is identical to $H$ in $T$
16: end if
```

Once a set of workloads are waiting to be mapped in the operating system task queue, the mapping procedure can be broken into three related steps: 1 search for the region candidates that meet the topology requirement within the context of RISO; 2) for each region candidate, verify the communication isolation criteria; and 3) select the final candidate close to the target MC.

Step 1 (Topology Searching): The algorithm will firstly search the NoC for proper candidates to serve the incoming workload. As explained in Section III-A, topology vector H, V of each workload are already attached by the OS. It informs the mapping algorithm the preferred topology and internal thread organization. If the number of free cores in F is fewer than that the application requires (number of $\langle t_i, p_j \rangle$ tuples in S), the searching process returns directly with a failure. Otherwise, it tries to find all possible candidates in the mesh. The detail of topology searching is described in Algorithm 1. For simplicity, it only shows the case that the shape is a reversed L and depicted by H. For other cases, the searching procedure is similar.

Lines 3–15 are responsible for searching the target shape denoted by H (horizontal vector in this example). The algorithm starts searching T row-by-row (line 4) 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, a target topology candidate is found (lines 4–13). Otherwise, if the algorithm has traversed all nodes in set F but still does not find shape identical to H (line 15), the algorithm returns with a failure. Note that Algorithm 1 is not limited to searching L shape, and to further boost consolidation density, it is also applicable to other shapes shown in Fig. 6.

Step 2 (Performance Verification): After successfully finding the target topology, we need to verify the performance isolation criteria, represented by the constraint that the aggregated link utilization of every shared link U_l will not exceed U_{congest}. First, we need to figure out the shared link set L under DOR routing mechanism. A link is regarded as a shared link if it bears the traffic of two different workloads. The aggregate traffic may be either core-to-core or memory access. Performance isolation must be strictly guaranteed for each shared link. As an example, Fig. 8 shows the hypothetical scenario after mapping

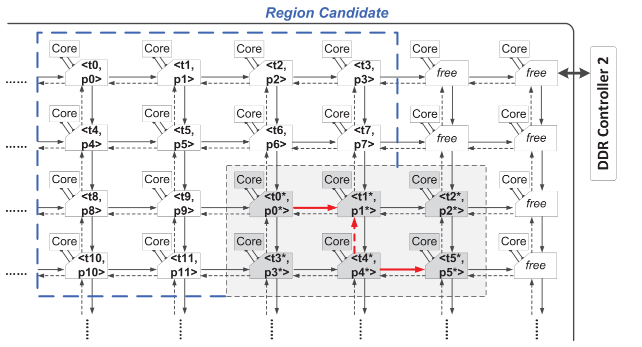


Fig. 8. Performance verification for a region candidate.

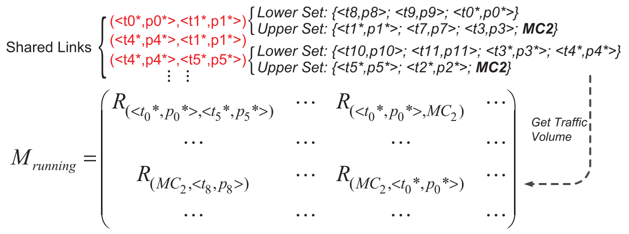

Shared Links
(<t0*,p0*>,<t1*,p1*>)
(<t4*,p4*>,<t1*,p1*>)
(<t4*,p4*>,<t5*,p5*>)
Lower Set: {<t8,p8>; <t9,p9>; <t0*,p0*>}
Upper Set: {<t1*,p1*>; <t7,p7>; <t3,p3>; MC2}
Lower Set: {<t10,p10>; <t11,p11>; <t3*,p3*>; <t4*,p4*>}
Upper Set: {<t5*,p5*>; <t2*,p2*>; MC2}
Get Traffic Volume
M_running =
R(<t0*,p0*>,<t5*,p5*>)
R(<t0*,p0*>,MC2)
R(MC2,<t8,p8>)
R(MC2,<t0*,p0*>)

Fig. 9. Shared link set identification.

Fig. 7(a). Every node is assigned by a $< t, p >$ tuple. Links can also be represented by tuple pairs as $(< t_{\text{from}}, p_{\text{from}} >, < t_{\text{to}}, p_{\text{to}} >)$. For example, we select three shared links under DOR routing mechanism in Fig. 8, and they can be presented as: $(< t_0^*, p_0^* >, < t_1^*, p_1^* >)$, $(< t_4^*, p_4^* >, < t_1^*, p_1^* >)$, and $(< t_4^*, p_4^* >, < t_5^*, p_5^* >)$, respectively.

Second, to calculate the shared link utilization, we must identify the nodes by which the injected traffic will use this shared link. Therefore, we divide the associate nodes into two sets: 1) lower_set and 2) upper_set, defined by (1). Obviously, a shared link will only carry the traffic generated from nodes in lower_set and terminated at nodes in upper_set, under DOR. Fig. 9 gives these node sets of the three shared links. In particular, Fig. 9 also regard MC as a destination or source, in order to bring memory request or response traffic into performance isolation verification. By looking up the traffic matrix M_{running}, we can obtain the overall flits that use a particular shared link

$$\begin{cases} \text{lower}_{\text{set}} : \{\text{Node} < t, p > \mid t = t_{\text{from}}\, p \le p_{\text{from}}\} \\ \text{upper}_{\text{set}} : \{\text{Node} < t, p > \mid t \le t_{\text{to}}\, p = p_{\text{to}}\}. \end{cases} \quad (1)$$

Algorithm 2 shows in detail the verification process. Lines 4–8 determine the aggregated traffic volume on each shared link. U is calculated in line 9. line 11 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 14 describes.

Step 3 (Near MC Mapping): The valid region candidates returned by Step 2 all guarantee communication isolation. RISO further reduces end-to-end memory access latency by adding Step 3. It will select the final candidate based on workload memory access distribution. In memory access vector MC, the item with the maximum value (line 1 in Algorithm 3) is regarded as the most visited MC.

To calculate the distance from a region candidate to this chosen MC, we use region distance as the metric. It is defined

Algorithm 2 Performance Verification

```
Input: Candidates: $candidates$; the traffic matrix: $M_{running}$; shared link set: $L$;
    time interval: $time$; link utilization threshold: $U_{congest}$;
Output: $valid\_candidates$; //after verification;
 1: for each $candidate$ in $candidates$ do
 2:     int $sum = 0$;
 3:     for each shared link $l(< t_{from}, p_{from} >, < t_{to}, p_{to} >) \in L$ do
 4:         for each $node < t_i, p_i > \in lower\_set$ of $l$ do
 5:             for each $node < t_j, p_j > \in upper\_set$ of $l$ do
 6:                 $sum$ += $R_{(<t_i,p_i>,<t_j,p_j>)}$; //sum up the values in traffic matrix
 7:             end for
 8:         end for
 9:         $U = \frac{sum}{time}$;
10:         if $U > U_{congest}$ then
11:             print "$invalid$"; //violate communication isolation
12:         end if
13:     end for
14:     $valid\_candidates$.push_back($candidate$); //$U$ of every shared link is lower
        than $U_{congest}$, communication isolation is ensured
15: end for
```

Algorithm 3 Near-MC Mapping

```
Input: Valid candidates: $valid\_candidates$; MC vector: $MC$;
Output: Final candidate: $can\_final$;
1: $mc = \max(MC)$; //most visited MC
2: for each $candidate \in valid\_candidates$ do
3:     for each $< t_i, p_j > \in candidate.H$ do
4:         $dist+ = D(< t_i, p_j >, mc)$
5:     end for
6:     $all\_dist$.push_back(avg($dist$)); //region distance
7: end for
8: $index = \min(all\_dist)$; //get final candidate
9: $can\_final = valid\_candidates[index]$;
```

as the average distance from each node in the region to the MC (lines 4–6). The final candidate is hence selected with the minimum region distance. Lines 8–9 show such procedure.

C. Traffic Prediction

As many prior NoC flow control techniques, RISO relies on accurate traffic prediction as an important guide to the application mapping. Before a preferred topology is mapped into the cloud processor, RISO predicts communication traffic and thereby identifies sharable links, based on historical values stored in (M_{running}).

Most prior work uses liner predictor to fulfill this purpose. We find that although liner predictor is capable for gradually changed traffic patterns, it is 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. In particular, we modified last value predictor (LVP) [23], [24] to handle both bursty and nonbursty scenarios. The prediction function is defined as

$$T_{\text{prediction}} = \begin{cases} h_2 & \mid \frac{(h_2 - h_1)}{h_1} \mid < \Omega \\ +\infty & \text{Otherwise.} \end{cases} \tag{2}$$

The predictor stores two most recent traffic volumes as h_1, h_2 for every source–destination pair. If the two values differ sharply (larger than a predefined threshold Ω), we exclude the associate links from sharable links by assigning a $+\infty$ to the final prediction value; otherwise, we still follow the LVP.

TABLE I
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

IV. Experimental Setup

We intend to evaluate RISO in two aspects: 1) system level, which evaluates the consolidation density from the whole system point of view and 2) network performance, which explores the benefit of RISO to network performance by tackling the routing-density tradeoff. First, we introduce the various metrics used in the experiments, and then, the performance evaluation setup and state-of-the-art baselines are specified.

A. Consolidation Density Metric

We use system utilization (U_{system}) [15] as one of the metrics for consolidation density evaluation. It represents the busy-cycle proportion of all cores in a cloud processor. A higher system utilization means more workloads have been mapped into the processor, and thus indicates a higher consolidation density. In particular, for a N-node system during T period of time, U_{system} is defined by

$$U_{\text{system}} = \frac{\sum_{i=1}^{N} T_i}{N \times T} \tag{3}$$

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 [15], which is defined by

$$\text{Load} = \frac{R \times S}{N \times I} \tag{4}$$

where R is average requested resources (i.e., cores in this paper) of all applications, I is average interarrival time between consecutive applications, and 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 I.

B. Performance Simulation Setup

We modified Booksim2.0 [25] to evaluate network performance. We select 13 benchmark programs from Parsec [16] and Cloudsuite [1], each of which acts as a workload. We run their traces obtained from two full system simulators, GEMS [26] for Parsec and Flexus [27] plus Simics

TABLE II
FULL SYSTEM SIMULATOR CONFIGURATION

Parameter	Value
Cloud Processor	4x4 in-order cores
Coherence Protocol	MOESI
L1 I/D Cache	32KB (2-way)
L1 Cache Access Latency	1 cycle
Shared L2 Cache	256KB/bank (4-way), 32MSHRs
L2 Cache Access Latency	8 cycles
Main Memory	16 GB DDR3, max 16 REQs/core
Memory access latency	160 cycles
Memory Controller	4MCs, placed at corners
Workloads (Parsec, Cloudsuite)	`blackscholes`, `fluidanimate`, `streamcluster`, `swaptions`, `x264`, `vips`, `freqmine`, `bodytrack`; `web_serving`, `data_analytics`, `data_serving`, `media_streaming`, `graphic_analytics`;

for Cloudsuite. The detailed configuration is shown in Table II. We collected both core-to-core and memory access traffic generated from all cores of each workload.

The NoC topology for our trace-driven simulation is a 8 × 8 mesh. The router is configured with a two-stage pipeline plus one cycle for link traversal. We use two virtual channels (VCs) and each has an 8-flit buffer. For different NoC configurations, the congestion threshold (U_{congest}) might be different. Therefore, we choose the most conservative value, 65%, in accordance with the results shown in Fig. 4. We also set the burstiness detection threshold [Ω in (2)] to 10% for the performance evaluation, which ensures the performance isolation under bursty scenarios. However, other design choices are also applicable, and Section V-C will give a thorough evaluation of Ω and its impact on performance isolation.

C. Baselines Compared

We compare RISO with three previously proposed schemes. The first one employs efficient routing but at the expense of lower consolidation density [28], denoted by regularity-oriented scheme [Fig. 1(a)]. The second scheme takes the opposite, i.e., emphasizing the density, but paying for more complex routing mechanisms [15], denoted by density-oriented scheme [Fig. 1(b)]. Besides, VCs could also be used to isolate traffic of different workloads, as shown in some state-of-the-arts [29]. Such scheme does not require a workload to be consolidated into a region with specific shapes, so it could obtain a near optimal system utilization. However, the application traffic shares the on-chip routers and physical links, so the interference still exists. We denote such scheme as VC oriented. In Section V, we compare RISO with these baselines to represent its efficacy in workload consolidation.

V. RESULTS AND ANALYSIS

A. Consolidation Density

Fig. 10 shows the system utilization from underload (x-axis before 1) to overload (x-axis after 1). The result shows that RISO improves the system utilization by up to 12% (16 × 16 mesh) higher than regularity-oriented scheme in the overload condition. Surprisingly, RISO performs almost equally well to density-oriented scheme (within 0.1% in 32 × 32 mesh). Even though RISO cannot exploit all irregular regions due to the link utilization constraint, it can

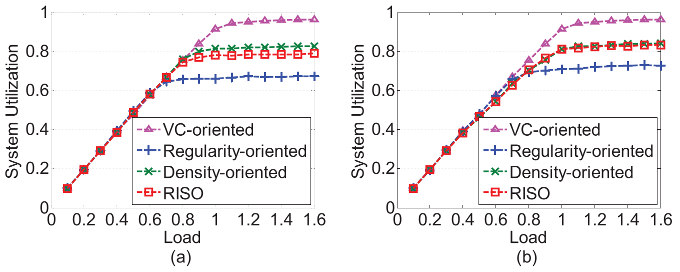


Fig. 10. System utilization for (a) 16 × 16 and (b) 32 × 32 mesh.

deal with some unique regions such as ⊏, which cannot be supported in density-oriented scheme, and enables our scheme to match density-oriented scheme in consolidation density. Compared to the VC-oriented approach, RISO has a 15% less consolidation density, because VC-oriented approach distributes the workload threads to cores randomly, which maximumly diminishes the fragmentation and makes it more possible to approach the theoretical upper bound of the system utilization.

B. Network Performance

Besides consolidation density, network-related parameters also impact the efficiency of workload consolidation. In this section, we elaborate in detail the benefits of RISO to network performance. First, we prove that RISO complies with performance isolation; then, using network latency as the performance metric, we show that RISO can significantly improve the overall NoC performance as well.

1) Performance Isolation Analysis: As described in Section II, regularity-oriented approach, though only applicable to core-to-core traffic, are regarded for ideal performance isolation by enforcing strict isolation. Hence, we compare VC-oriented and RISO with regularity-oriented scheme to verify their capability of performance isolation, and the results are shown in Table III. The three columns in the center show the actual latency values of various benchmarks under three schemes. The last two columns show the latency variation normalized to results of regularity-oriented approach. As can be seen, RISO exhibits an average latency variation within 1.8×10^{-4} compared with regularity-oriented approach; in contrast, VC-oriented approach shows a 31.7% and 33.5% latency degradation for Parsec and Cloudsuite, respectively, because of its interworkload interference. This experiment proves that for VC-oriented approach, performance isolation cannot be preserved under limited VCs. However, RISO could completely guarantee the performance isolation.

2) Core-to-Core Communication: Density-oriented scheme is notorious for its network performance, although it can enforce performance isolation and provide high consolidation density. Hence, we show how much latency improvements can be achieved by RISO. We compare RISO to density- and VC-oriented approach in this experiment. Note that RISO and VC-oriented approach use efficient DOR, while density-oriented scheme uses the most favorable Up*/Down* [12] routing mechanism.

To clearly illustrate the routing influence to network performance, we evaluate the network latency using workload

TABLE III
PERFORMANCE ISOLATION RESULTS (CORE-TO-CORE TRAFFIC ONLY)

Workloads	Regularity -oriented	RISO	VC -oriented	Latency Variation RISO	Latency Variation VC-ori.
blackscholes	22.433914	22.414153	29.874921	8.82×10^{-04}	3.33×10^{-01}
fluidanimate	11.752759	11.753138	15.434534	3.22×10^{-05}	3.13×10^{-01}
streamcluster	11.599542	11.599961	16.393874	3.61×10^{-05}	4.13×10^{-01}
swaptions	13.311812	13.30962	19.273942	1.65×10^{-04}	4.48×10^{-01}
x264	14.503931	14.500583	18.123512	2.31×10^{-04}	2.50×10^{-01}
vips	12.506349	12.506229	15.618726	9.60×10^{-06}	2.49×10^{-01}
freqmine	11.268146	11.267128	14.126142	9.04×10^{-05}	2.54×10^{-01}
bodytrack	17.315179	17.315094	22.124364	4.91×10^{-06}	2.78×10^{-01}
web_serving	24.233341	24.235427	33.348398	8.61×10^{-05}	3.76×10^{-01}
data_analytics	22.697842	22.697993	29.384938	6.65×10^{-06}	2.95×10^{-01}
data_serving	18.293838	18.297653	25.287348	2.08×10^{-04}	3.82×10^{-01}
media_streaming	25.458476	25.45529	32.894345	1.25×10^{-04}	2.92×10^{-01}
graphic_analytics	14.480122	14.484204	19.287394	2.82×10^{-04}	3.32×10^{-01}

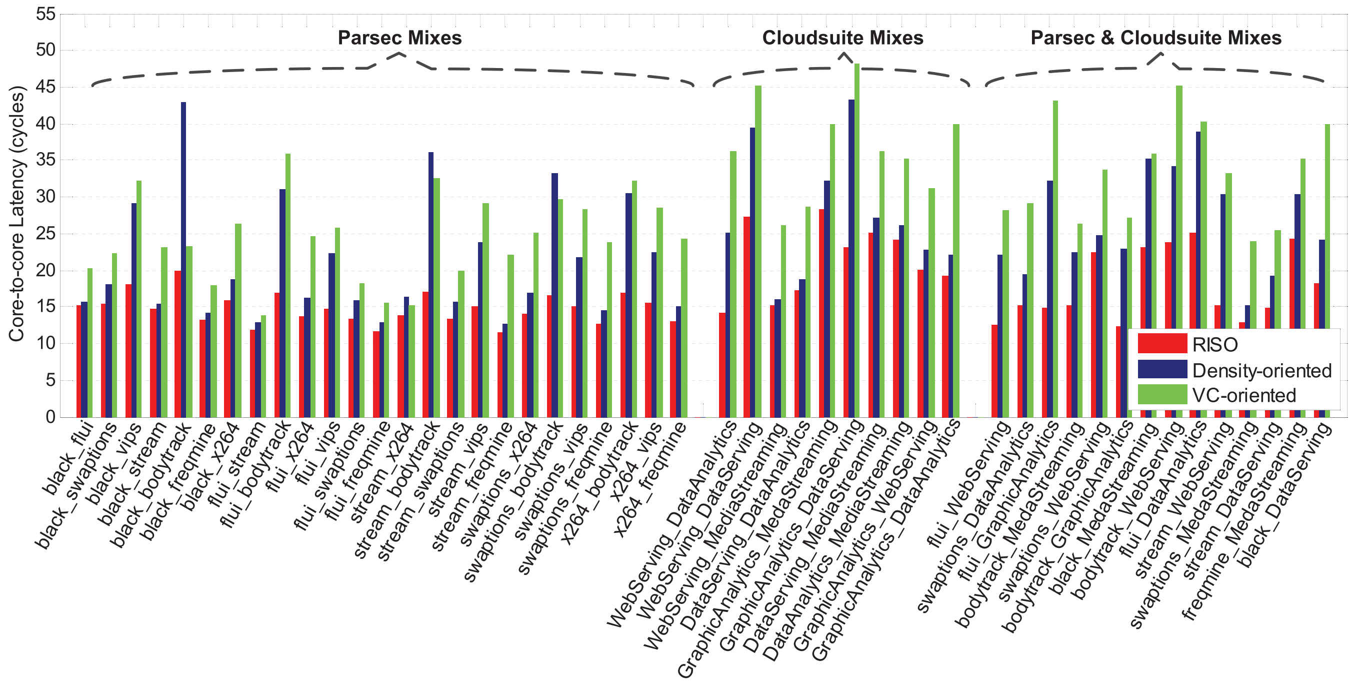


Fig. 11. Average core-to-core latency comparison using Parsec and Cloudsuite workload mixes.

mixes, and each mix only consists of two workloads randomly selected from Parsec and Cloudsuite (not 13 workloads as a whole). Fig. 11 shows the results of three schemes for 49 workload mixes. We categorize these mixes into three classes. Clearly, RISO wins for all. Generally speaking, VC-oriented approach, even if using DOR, exhibits more severe latency degradation than density-oriented approach, which means that the contention delay is the major factor compared with the routing delay. However, for some mixes, i.e., `black_bodytrack` and `swaptions_bodytrack`, routing delay is the chief dictator. RISO uses efficient routing and simultaneously eliminates interference, thus providing 40.2% and 73.2% latency improvements averaged by the three classes.

3) *Memory Access:* Memory access traffic cannot be isolated by any previously proposed approaches, because the memory traffic will inevitably traverse into other workload's territory to reach the desired MC. However, the numerous shared links along the path to the MC could be uniquely exploited by RISO. RISO relies on the verification of shared link utilization that provides the opportunity for memory access traffic to proceed in harmony with other interworkload traffic, avoiding the interference. Fig. 12 justifies this concept. We run Parsec and Cloudsuite benchmark programs under RISO and regularity-oriented approach. Obviously, it confirms that the regularity-oriented approach cannot isolate memory access traffic. Even if we augment it with near-MC mapping as implemented in the RISO (denoted as regularity-oriented + near-MC in the figure), the interference still exists, because the shared link utilization may also exceed the safety threshold that is ignored by these approaches. In contrast, RISO is utilization aware, and the mapping is only applicable if link sharing allows, which conduces a 7.1% and 15.9% average latency improvement for Parsec, and 14.7%, 28.2%

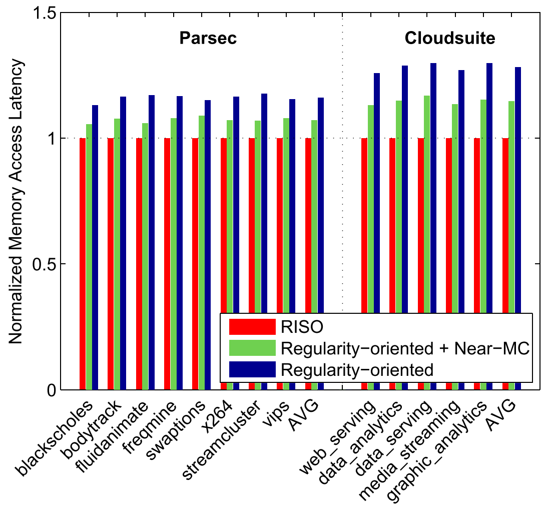


Fig. 12. Average memory access latency comparison. Regularity-oriented approach cannot isolate memory access traffic even if near-MC mapping is integrated, and the memory traffic could also be blocked by other workload's traffic.

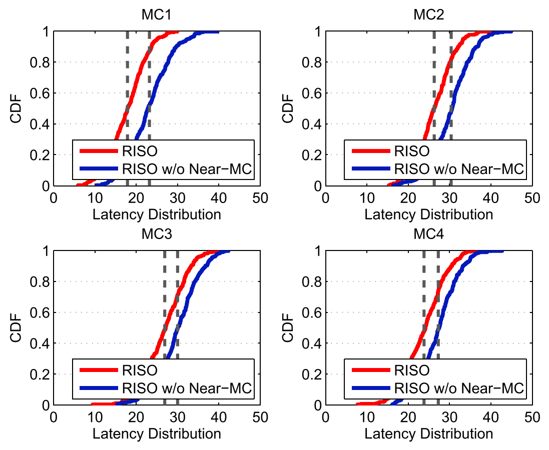


Fig. 13. CDF of RISO memory access latency with/without near-MC mapping implemented.

for Cloudsuite. This experiment proves that the sole employment of near-MC mapping is not enough to isolate memory access traffic. Shared link utilization is the utmost consideration that needs to be enforced.

4) Near-MC Mapping Analysis: To quantify the benefits of near-MC mapping, we study the memory access latency within the context of RISO under two scenarios: 1) with near-MC mapping implemented and 2) without near-MC mapping implemented. In this set of experiment, we trace 256 memory access packets for each MC. The latency of each packet is recorded and plotted using cumulated distribution function (cdf), as shown in Fig. 13. Two vertical lines represent the latency median (50% point at y-axis) of the recorded packets under RISO and RISO without near-MC, respectively. For all the four MCs, the near-MC mapping reduces memory access latency substantially, i.e., 10.3% for MC1 as the maximum and 7.6% for MC3 as the minimum, due to a shortened physical distance between a workload and its favorable MC.

C. Predictor Robustness

RISO relies on the LVP to guarantee the performance isolation requirement, which involves two aspects: 1) traffic value prediction and 2) the detection of traffic burstiness. At the worst case scenario, the traffic pattern of a workload cannot be acquired in advance, so the robustness of RISO must be evaluated to explore the least time that must be spent to achieve the correct prediction. We vary the prediction interval as 0.5, 1, and 2 ms, and calculate the precision that is defined as the differentiation of the predicted value and real value. Fig. 14 shows the precision under the three scenarios, which is categorized into four ranks. Taking the rank 0%–2% at 1 ms as an example, the precision can attain 98.2% for Parsec and 86.3% for Cloudsuite. Compared to 0.5-ms scenario, the average precision improvement is 2.6%. However, the precision is almost the same under 1- and 2-ms interval (<0.5% difference). It means that even if we do not have any knowledge in terms of workload traffic pattern, 1–2 ms interval would be enough for RISO to make correct predictions. Moreover, modern operating system kernel issues process scheduling at <100-ms magnitude [30], which indicates that RISO is able to finish all of its computation and prediction procedure at optimally 1% of OS scheduling time, and is also tiny enough to be deployed in modern operating systems.

For the mispredictions, i.e., rank ±10%, it indicates that the traffic is bursty and must be carefully handled to preserve performance isolation, especially for Cloudsuite benchmarks that the user incoming requests are stochastic. In RISO, predefined threshold Ω is used to filter out the bursty traffic from shared links. Fig. 15 shows the shared link utilization tuned by the value of Ω, under 1-ms prediction interval. As shown in the figure, Ω could be scaled larger without violating the 65% link utilization threshold for most of the benchmarks. Larger Ω indicates a larger consolidation density, with performance isolation well guaranteed at the same time. However, setting Ω naively large is not always beneficial. For example, benchmarks, such as `graphic_analytics` and `data_analytics`, will be interfered if Ω is set larger than 12%. Generally speaking, 10% is a safe value that brings both high consolidation density and guaranteed performance,

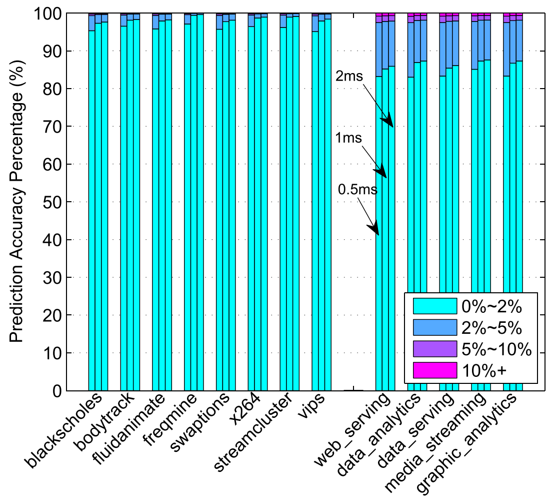


Fig. 14. Prediction accuracy of last level prediction for Parsec and Cloudsuite benchmarks. The calculation is issued under 1-ms interval and runs through the whole execution.

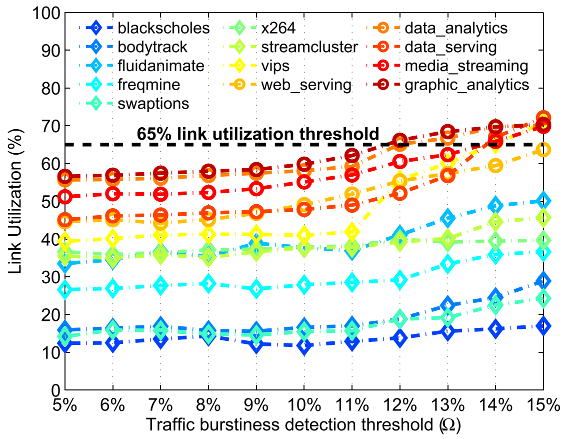


Fig. 15. Traffic burstiness threshold impact on performance isolation criterion.

and that is also why we use this value for network performance evaluation in Section V-B.

VI. Related Work

First, it was introduced by Flich *et al.* [14] that the performance isolation technique is imperative to achieve controllable QoS in NoC. 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 which 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 [28]. 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.* [15] proposed an irregular-shaped isolation based on complex routing mechanism. This method also follows strict isolation between workloads, and improves consolidation density compared with rectangle-based isolation. However, its routing mechanism is less efficient and exhibits substantial degradation with respect to network performance.

For the optimization of the memory access latency, Das *et al.* [19] proposed an application-to-core mapping policies to improve system performance by reducing interapplication interference. By mapping network-sensitive workloads close to the MC, memory access latency could be diminished. Sharifi *et al.* [31] proposed two packet prioritization schemes, which cooperatively improve network performance by reducing end-to-end memory access latency. However, these two techniques can only apply to single-threaded workloads (i.e., SPEC CPU 2006 [32]), so they ignore the core-to-core communication interference inherited from multithreaded workloads. RISO takes both core-to-core and memory access traffic into account, and performance isolation is truly enforced in NoC.

VII. Conclusion

This paper proposes the RISO strategy to enforce performance isolation in cloud processors. Unlike traditional strict isolation strategy, such as regularity-oriented and density-oriented approach, RISO allows underutilized links to be shared by multiple applications, as long as the aggregate link utilization is lower than a certain 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 cost-effective DOR and hence yields higher network performance. In other words, RISO effectively resolves the tradeoff between consolidation density and network performance. For the memory access traffic that strict isolation strategies fail to isolate, the resource sharing in RISO also provides a unique opportunity to eliminate the memory access interference. With the help of near-MC mapping method, workloads can even benefit from the accelerated memory access. We therefore believe that RISO is a promising scheme for workload consolidation in many-core cloud processors.

References

[1] M. Ferdman *et al.*, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proc. 17th Int. Conf. Archit. Support Program. Lang. Operat. Syst.*, 2012, pp. 37–48.

[2] P. Lotfi-Kamran *et al.*, "Scale-out processors," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, 2012, pp. 500–511.

[3] *Intel Single-Chip Cloud Computer*. [Online]. Available: http://www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-article.html, accessed Mar. 2013.

[4] *AMD Opteron 6000+ Series*. [Online]. Available: http://www.amd.com/en-us/products/server/opteron/6000, accessed Jun. 2013.

[5] *Amazon Elastic Cloud Computing*. [Online]. Available: http://aws.amazon.com/ec2/, accessed May 2010.

[6] M. R. Marty and M. D. Hill, "Virtual hierarchies to support server consolidation," in *Proc. 34th Annu. Int. Symp. Comput. Archit.*, 2007, pp. 46–56.

[7] H. Lu, G. Yan, Y. Han, B. Fu, and X. Li, "RISO: Relaxed network-on-chip isolation for cloud processors," in *Proc. 50th IEEE Annu. Design Autom. Conf.*, May/Jun. 2013, pp. 1–6.

[8] A. K. Mishra, O. Mutlu, and C. R. Das, "A heterogeneous multiple network-on-chip design: An application-aware approach," in *Proc. 50th Annu. Design Autom. Conf.*, 2013, pp. 1–10.

[9] S. Volos, C. Seiculescu, B. Grot, N. K. Pour, B. Falsafi, and G. De Micheli, "CCNoC: Specializing on-chip interconnects for energy efficiency in cache-coherent servers," in *Proc. 6th IEEE/ACM Int. Symp. Netw.-Chip*, May 2012, pp. 67–74.

[10] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a Teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep./Oct. 2007.

[11] S. Bell *et al.*, "TILE64—Processor: A 64-core SoC with mesh interconnect," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2008, pp. 88–89.

[12] M. D. Schroeder *et al.*, "Autonet: A high-speed, self-configuring local area network using point-to-point links," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 8, pp. 1318–1335, Oct. 1991.

[13] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie, "Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori," in *Proc. 20th Int. Parallel Distrib. Process. Symp.*, Apr. 2006, pp. 10–19.

[14] J. Flich *et al.*, "On the potential of NoC virtualization for multicore chips," in *Proc. Int. Conf. Complex, Intell. Softw. Intensive Syst.*, 2008, pp. 801–807.

[15] A. 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 Perform. Comput.*, 2007, pp. 500–513.

[16] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. Compil. Techn.*, 2008, pp. 72–81.

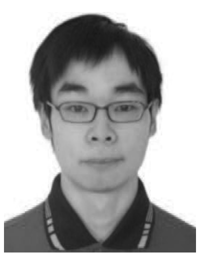
[17] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *Proc. 9th Int. Symp. High-Perform. Comput. Archit.*, 2002, pp. 91–102.
[18] J. W. van den Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-controlled best-effort communication for networks-on-chip," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, 2007, pp. 1–6.
[19] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi, "Application-to-core mapping policies to reduce memory system interference in multi-core systems," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit.*, Feb. 2013, pp. 107–118.
[20] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. 50th IEEE Annu. Design Autom. Conf.*, May/Jun. 2013, pp. 1–10.
[21] M. Kandemir, O. Ozturk, and S. P. Muralidhara, "Dynamic thread and data mapping for NoC based CMPs," in *Proc. 46th IEEE Annu. Design Autom. Conf.*, Jul. 2009, pp. 852–857.
[22] B. Fu, Y. Han, J. Ma, H. Li, and X. Li, "An abacus turn model for time/space-efficient reconfigurable routing," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, 2011, pp. 259–270.
[23] Y. S.-C. Huang, K. C.-K. Chou, C.-T. King, and S.-Y. Tseng, "NTPT: On the end-to-end traffic prediction in the on-chip networks," in *Proc. 47th ACM/IEEE Annu. Design Autom. Conf.*, Jun. 2010, pp. 449–452.
[24] Y. S.-C. Huang, K. C.-K. Chou, and C.-T. King, "Application-driven end-to-end traffic predictions for low power NoC design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 2, pp. 229–238, Feb. 2013.
[25] *Booksim2.0*. [Online]. Available: https://nocs.stanford.edu/, accessed Jan. 2012.
[26] *GEMS Simulator*. [Online]. Available: http://research.cs.wisc.edu/gems/publications.html, accessed May 2011.
[27] *Flexus Simulator*. [Online]. Available: http://parsa.epfl.ch/simflex/flexus.html, accessed Oct. 2013.
[28] V. Gupta and A. Jayendran, "A flexible processor allocation strategy for mesh connected parallel systems," in *Proc. Int. Conf. Parallel Process.*, 1996, pp. 166–173.
[29] F. Trivino, J. L. Sanchez, F. J. Alfaro, and J. Flich, "Exploring NoC virtualization alternatives in CMPs," in *Proc. 20th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, 2012, pp. 473–482.
[30] W. R. Stevens and S. A. Rago, *Advanced Programming in the UNIX Environment*. Reading, MA, USA: Addison-Wesley, 2005.
[31] A. Sharifi, E. Kultursay, M. Kandemir, and C. R. Das, "Addressing end-to-end memory access latency in NoC-based multicores," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2012, pp. 294–304.
[32] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

Hang Lu received the B.Eng. and M.Eng. degrees from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2008 and 2011, respectively. He is currently pursuing the Ph.D. degree in computer architecture with the University of Chinese Academy of Sciences, Beijing.

His current research interests include high-performance networks-on-chip, power efficient manycore architectures, and scale-out processors.

Binzhang Fu (M'09) received the B.Eng. degree in electronics and information engineering, and computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2004, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2011.

He is currently an Associate Professor with ICT, CAS. His current research interests include high-performance and high-reliable interconnection networks.



Ying Wang (S'11) received the B.Eng. and M.Eng. degrees in electrical engineering from the Harbin Institute of Technology, Harbin, China, in 2007 and 2009, respectively. He is currently pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

His current research interests include reconfigurable computing, interconnects, memory system, and fault-tolerance for many-core architectures.

Yinhe Han (M'06) received the B.Eng. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2001, and the M.Eng. and Ph.D. degrees in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2003 and 2006, respectively.

He is currently a Professor with the State Key Laboratory of Computer Architecture, ICT, CAS. His current research interests include computer architecture, in particular, on fault-tolerant and low-power architecture and VLSI design and test.

Prof. Han is a member of the Association for Computing Machinery/China Computer Federation/Institute of Electronics, Information and Communication Engineers. He was a recipient of the Best Paper Award at the Asian Test Symposium (ATS) in 2003. He was the Program Chair of ATS in 2014, and was the Finance Chair of the International Symposium on High-Performance Computer Architecture (HPCA) in 2013 and the Program Co-Chair of the Workshop on RTL and High Level Testing in 2009. He served on the Technical Program Committees of multiple IEEE and ACM conferences, including the International Conference on Parallel Architectures and Compilation Techniques (PACT) in 2014, HPCA in 2013, the Asia and South Pacific Design Automation Conference (ASPDAC) in 2013, Cool Chip in 2013, ATS from 2008 to 2010, and Great Lakes Symposium on VLSI from 2009 to 2010.

Guihai Yan (M'11) received the B.Sc. degree in electronics and software engineering from Peking University, Beijing, China, in 2005, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2011.

He is currently an Associate Professor with ICT, CAS. His current research interests include computer architecture, domain-specific microsystems, and energy-efficient computing.

Xiaowei Li (SM'04) received the B.Eng. and M.Eng. degrees from the Hefei University of Technology, Hefei, China, in 1985 and 1988, respectively, and the Ph.D. degree from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1991, all in computer science.

He is currently a Professor and the Deputy Director of the Key Laboratory of Computer System and Architecture with ICT, CAS. His current research interests include VLSI testing and design verification, dependable computing, and wireless sensor networks.

Prof. Li is an Associate Editor-in-Chief of the *Journal of Computer Science and Technology* and a member of the Editorial Board of the *Journal of Electronic Testing* and the *Journal of Low Power Electronics*. He serves on the Technical Program Committees of multiple IEEE and ACM conferences, including the Vehicular Technology Society, the Design Automation and Test in Europe, the Asia and South Pacific Design Automation Conference, and the Conference on Dependable Computing. He was the Program Co-Chair of the IEEE Asian Test Symposium (ATS) in 2003, and the General Co-Chair of ATS in 2007.