Defect Tolerance in Homogeneous Manycore Processors Using Core-Level Redundancy with Unified Topology

Lei Zhang[†], Yinhe Han[†], Qiang Xu[‡], and Xiaowei Li[†*]

[†]Key Laboratory of Computer System and Architecture
Institute of Computing Technology, Chinese Academy of Sciences
{zlei, yinhes, lxw}@ict.ac.cn

[‡]Department of Computer Science & Engineering
The Chinese University of Hong Kong
qxu@cse.cuhk.edu.hk

Abstract

Homogeneous manycore processors are emerging for tera-scale computation. Effective defect tolerance techniques are essential to improve the yield of such complex integrated circuits. In this paper, we propose to achieve fault tolerance by employing redundancy at the core-level instead of at the microarchitecture-level. When faulty cores existing on-chip in this architecture, how to reconfigure the processor with the most effective topology is a relevant research problem. We present novel solutions for this problem, which not only maximize the performance of the manycore processor, but also provide a unified topology to operating system and application software running on the processor. Experimental results show the effectiveness of the proposed techniques.

1. Introduction

As technology advances, industry has started to employ multiple cores on a single silicon die. Significant research has been undertaken on tera-scale computing that is able to integrate tens to hundreds of homogeneous processor cores on a single chip to process massive amounts of information in parallel [1, 2, 3]. For example, an 80-core teraflop processor prototype was demonstrated at Intel Developer Forum 2006. Such processors containing a large number of cores are called manycore processors (note the difference from multicore processor that contains a small number of cores). In terms of communication infrastructure, Network-on-Chip (NoC) is generally regarded as the most promising interconnect solution for such giga-scale ICs [4], in which the topology determines the ideal performance of the on-chip network whereas the routing algorithm and the flow control mechanism determine how much of this potential is realized.

There are many challenges for the architecture design of manycore processors, in which manufacturing yield is one of the most serious concerns because an IC's profitability depends heavily on it [6]. With the ever-increasing circuit density, obtaining high fabrication yield solely through improving the manufacturing process is increasingly difficult and will become unaffordable in the near future. A more practical solution is to provide defect tolerance capabilities on-chip by incorporating redundant circuits. For example, memory Built-In-Self-Repair (MBISR) techniques have been widely utilized in the industry and prove to be very effective to keep the high fabrication yield of memory circuits. Such techniques should be extended to other types of VLSI circuits as well [5].

However, tolerating defects in processors is quite different from memory circuits because its internal structure is not as regular as memory cells. Previous attempts in this domain mainly focused on introducing microarchitecture-level redundancy (e.g., [7, 8]). This is appropriate for multicore processors (e.g., a quad-core processor) in order to keep the hardware overhead small. When the number of on-chip cores increases to a point that single core becomes inexpensive when compared to the entire manycore processor (e.g., a 64-core processor), however, it is not necessary to tolerate defective cores at the microarchitecture-level. Instead, it is more appropriate to employ core-level redundancy in such cases to reduce the complexity associated with microarchitecture-level redundancy.

With core-level redundancy, faulty cores are replaced by spare ones placed on-chip. Therefore, it is possible that the topology of the target design is modified and different fabricated chips may have different underlying topologies. This is a big burden for programmers because an optimized program for one topology may not work well for a different one and the programmers are facing various topologies when optimizing their parallel programs.

To address the above problem, the concept of logical topology is introduced in this paper. A logical topology is isomorphic with the topology of the target design but is a degraded version. From the viewpoint of the operating system (OS) and the programmers, they always see a unified logical topology regardless of the various underlying physical topologies. This eases the dispatching and scheduling tasks for OS and facilitates the optimization of parallel programs. To compare the performance of different logical topologies, we introduce two evaluation metrics, namely distance factor (*DF*) and congestion factor (*CF*). Effective topology reconfiguration techniques are then presented to find the best logical topology in terms of the two evaluation metrics. Experimental results on a hypothetical 64-core manycore processor show the effectiveness of the proposed techniques.

The rest of this paper is organized as follows. Section 2 presents the motivation of this work. In Section 3, we formulate the topology reconfiguration problem investigated in this paper. The proposed algorithm is then described in detail in Section 4. Next, Section 5 presents experimental results. Finally, we conclude this paper in Section 6.

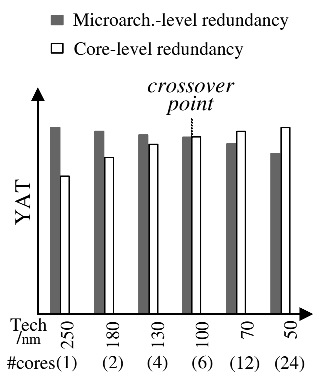
[*]To whom correspondence should be addressed.

The work of L. Zhang, Y. Han and X. Li was supported in part by National Natural Science Foundation of China (NSFC) under grant No.(60633060, 60606008, 60576031), in part by National Basic Research Program of China (973) under grant No. 2005CB321604, and in part by Hi-Tech Research and Development Program of China under grant No.2007AA01Z109. The work of Y. Han was also supported by the fund of Chinese Academy of Sciences (No. 20074010) due to the President Scholarship. The work of Q. Xu was supported in part by the Hong Kong SAR RGC Earmarked Research Grants 2150503 and 2150558.

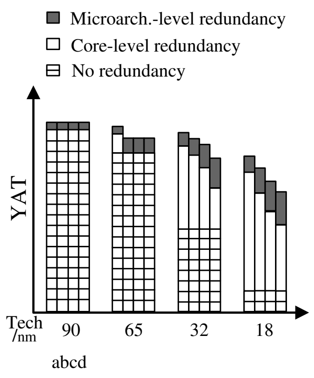
2. Motivation

2.1. Core-level redundancy

Researchers evaluate the effectiveness of various redundancy mechanisms using yield-adjusted throughput (YAT), which shows the average chip throughput when a large number of chips are fabricated [7, 8]. As can be observed from Fig 1(a), there is a crossover point, from which core-level redundancy will bring better YAT than microarchitecture-level redundancy. Similarly, in Fig 1(b), we can observe, as technology advances, YAT becomes increasingly lower without redundancy. At the same time, microarchitecture-level redundancy (grey part) brings YAT improvement, but at a smaller scale when compared to core-level redundancy (white part) in newer technology generation.



(a) redrawn from [7].



(b) redrawn from [8].

Figure 1. YAT comparison between microarchitecture-level and core-level redundancy.

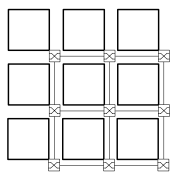
From the above, we can conclude that, for homogeneous manycore processors that contain a large number of on-chip cores and are fabricated in latest technology, providing spare cores on-chip is more beneficial when compared to providing microarchitecture-level redundancy.

There are two schemes to design homogeneous multicore processors or manycore processors with core-level redundancy, namely *As Many As Available* (AMAA) and *As Many As Demand* (AMAD). The AMAA scheme, adopted in Sun's UltraSPARC T1 processors [9], degrades a chip by disabling faulty cores only. For example, a fabricated quad-core processor can be a full version with 4 operational cores; or it can be degraded to a tri-core, dual-core or single-core processor depending on the number of cores that are faulty. In AMAD scheme, also denoted as "N+M" mechanism in this paper, an N-core processor is provided with M redundant cores and we always provide customers with N operational cores. That is, it is possible that there are fault-free cores left unused in AMAD scheme.

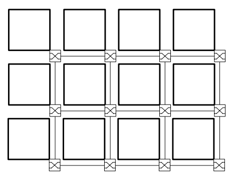
It is preferred to employ the AMAA scheme in multicore processors to keep the overhead small. However, as the number of on-chip cores increases, the overhead of leaving a few redundant cores on-chip unused is acceptable because a single core is inexpensive compared to the entire chip. In addition, with many cores implemented on-chip, we may get various types of degraded chips (with different number of faulty cores) after fabrication and the yield of the demanded N-core processor cannot be promised in AMAA scheme. Finally, from a commercial point of view, as there are many different degraded versions, it may cause some confusion in marketing. Therefore, for manycore processors, AMAD scheme is preferred and we mainly focus on this scheme in this paper.

2.2. Physical topology and logical topology

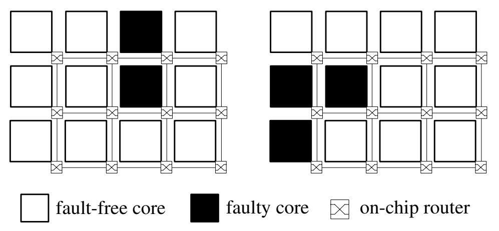
In homogeneous manycore processors, the performance of the on-chip communication infrastructure significantly affects the efficiency of parallel applications. In order to minimize the communication overhead among threads or tasks, today's OS relies on explicit knowledge of the underlying topology [13]. For example, in Microsoft Windows Server 2003, a so-called advanced configuration and power interface (ACPI) hardware is used to pass a description of the physical topology of the system to the OS [11]. The topology information is used by Windows when dispatching and scheduling tasks. Topology information is also provided to programmers through API functions to optimize application software [11].



(a) What we expect.



(b) What we implement.



(c) What we get.

Figure 2. Faulty cores change the topology of target design.

In AMAD scheme, as the cores that are fabricated to be defective are not known a priori, when they are replaced by spare cores, the topology of the target design can be different. For example, suppose we want to provide 9-core processors with 3×3 2-D mesh topology to customers, as shown in Fig 2(a). Also, suppose 3 redundant cores (1 column) are provided to improve the yield of these chips as shown in Fig 2(b). If some cores (no more than 3) are defective, we could still get 9-core processors. However, as shown in Fig 2(c), if faulty cores are replaced by spare cores, not only the topologies that we get are different from what we expect, but also the topologies of different chips are distinct. It would be rather cumbersome for OS and programmers to face many different topologies and optimize them differently.

To address the above problem, we propose the concept of logical topology and we try to provide a unified topology regardless of the underlying one. Before introducing the details, we first define *Reference Topology* as the topology of the target design that we expect. For example, the 3×3 2-D mesh topology in Fig 2(a) is the expected reference topology.

For the "9+3" manycore processor shown in Fig 2(b), suppose the 7^{th}, 10^{th} and 11^{th} cores are defective after fabrication as shown in Fig 3(a), these cores are considered to be removed out of the chip. The remaining fault-free cores and their interconnections construct a *Physical Topology* as shown in Fig 3(b). It should be emphasized that once a manycore processor is taped out, its physical topology is determined and cannot be changed during its lifetime. This is fundamentally different from board-level multiprocessor systems, which are much easier to be repaired since the target topology can be maintained by simply replacing the faulty processor with a good one.

Based on AMAD scheme, a 9-core processor can still be provided but with different topology when compared to the reference topology. That is, we can construct a *Logical Topology* of the chip based on the given physical topology, which is isomorphic with the reference topology. An example is shown in Fig 3(c), in which we construct a *logical* 3×3 2-D mesh topology.

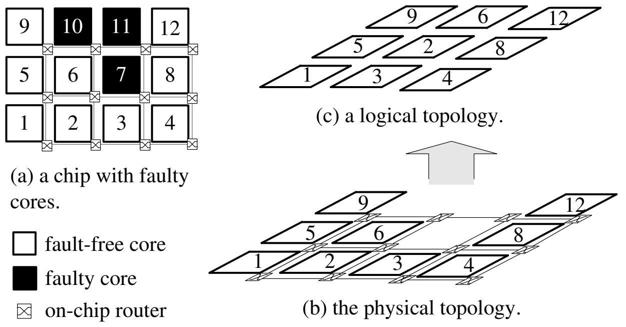


Figure 3. Physical topology and logical topology.

By introducing the concept of logical topology, OS and programmers always see a unified topology that is isomorphic with the reference topology, no matter how the underlying cores are connected physically. This greatly simplifies task dispatching and scheduling duties for OS and also facilitates the optimization of parallel programs. In addition, a unified topology that isolates various physical topologies for different chips also eases manycore processors' marketing process.

We are able to construct many logical topologies for a homogeneous manycore processor with a specific physical topology since any two fault-free cores can be logical neighbors to each other. A relevant research problem is then how to select the best one and provide it to the customers. This topology reconfiguration problem is quite different from MBISR, although both use redundant components to achieve yield improvement. In MBISR, the physical structures have to be maintained the same before and after reconfiguration, determined by the usage of memory chips [10]. However, in manycore processors, every core is an autonomous system and is able to communicate with other cores through on-chip interconnection network. The physical topology is therefore not necessary to be kept the same after reconfiguration. Only a unified logical topology needs to be maintained as described before.

It should be also noted that, there are many ways to implement the mapping from various physical topologies to their corresponding logical topologies for manycore processors. For example, one possible solution is to assign every core with a configurable index and fix it after fabrication to construct the expected logical topology.

3. Topology reconfiguration problem

A logical topology is typically a degraded version of the reference topology with faulty-cores on-chip. Since there can be many logical topologies for a particular physical topology, we should choose the one that has the best performance. In order to do so, two evaluation metrics are introduced in this section to model the performance degradation of different logical topologies when compared to the reference topology, namely distance factor (*DF*) and congestion factor (*CF*). It is important to note that the communication infrastructure is assumed to be fault-free in this research work.

Distance Factor: Distance factor is used to evaluate the communication delay between cores. The distance factor between two logical nodes n and n' ($DF_{nn'}$) is defined as the physical hops between them ($DF_{nn'} = Hops_{nn'}$). The distance factor of node n (DF_n) is defined as the average distance factor between node n and all its logical neighbors:

$$DF_n = \frac{1}{k}\sum_{n'=1}^{k} DF_{nn'}; \tag{1}$$

(Node n has k logical neighbors)

The distance factor of a logical topology (*DF*) is defined as the average DF_n of all nodes:

$$DF = \frac{1}{N}\sum_{n=1}^{N} DF_n; \tag{2}$$

(There are in total *N* nodes in the logical topology)

The reference topology has the minimum *DF* as all logical neighbors are usually located next to each other physically. For example, *DF* is 1 in mesh and torus topologies, which means that each pair of logical neighbors is exactly one hop away from each other. Larger value of *DF* means longer communication delay among logical neighbors.

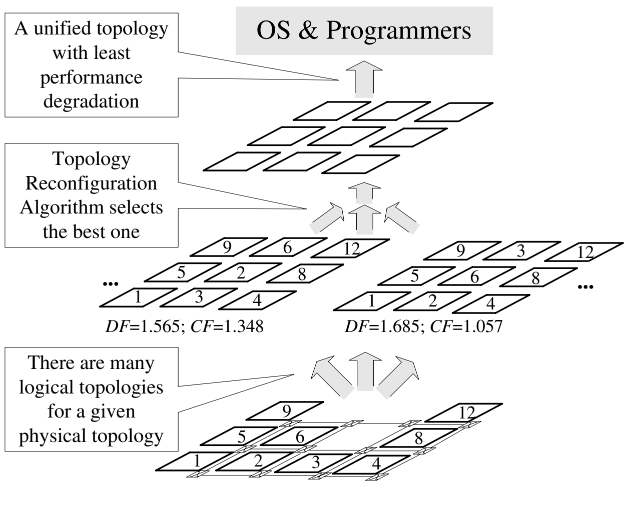
Congestion Factor: A logical topology not only changes the average distance among cores but also affects the distribution of traffic flows. Traffic may become uneven among different links. We define the congestion factor of a physical link *l*, denoted as CF_l as follows: for any nodes n and n', if they are logical neighbors, and *l* is on one of the routing paths between them according to the NoC's routing mechanism (e.g., XY-routing), we add CF_l by 1. For the reference topology, all links have the same congestion factors. However, for a degraded logical topology, some physical links may become congested with greater CF_l while the others with smaller CF_l.

Based on the above, we define the congestion factor of a logical topology (*CF*) as the standard deviation of CF_l of all links.

$$CF = \sqrt{\frac{\sum_{l=1}^{L}(CF_l - \overline{CF_l})^2}{L-1}}; \tag{3}$$

(There are totally *L* links in the physical topology)

CF of the reference topology is 0, which means that traffic can be evenly distributed across the network. Greater *CF* means less even flow distribution. Please note that even though advanced routing algorithms can be introduced to balance flow distribution, *CF* is able to evaluate the raw flow distribution which reflects the quality of a logical topology.



A unified topology with least performance degradation
OS & Programmers
Topology Reconfiguration Algorithm selects the best one
DF=1.565; CF=1.348
DF=1.685; CF=1.057
There are many logical topologies for a given physical topology

Figure 4. Topology reconfiguration.

With the above two metrics, the quality of different logical topologies can be evaluated and compared. *DF* and *CF* might be conflicted with each other during optimization, hence we unify them together. The Unified Metric (*UM*) is defined as

$$UM = w_{DF} \times DF + w_{CF} \times CF, \tag{4}$$

in which w_{DF} and w_{CF} are the optimization weights designated by users ($w_{DF} + w_{CF} = 1$).

Suppose the reference topology is mesh or torus, the topology reconfiguration problem investigated in this paper can be formulated as follows:

[**Topology Reconfiguration Problem (TRP)**] For a $N = R \times C$ homogeneous manycore processor with *M* redundant cores, suppose *D* cores ($D \leq M$) are faulty, construct $R \times C$ coordinates as follows:

$$\left[\begin{array}{cccc} (R-1,0) & (R-1,1) & \cdots & (R-1,C-1) \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ (1,\ 0) & (1,\ 1) & \cdots & (1,\ C-1) \\ (0,\ 0) & (0,\ 1) & \cdots & (0,\ C-1) \end{array}\right]$$

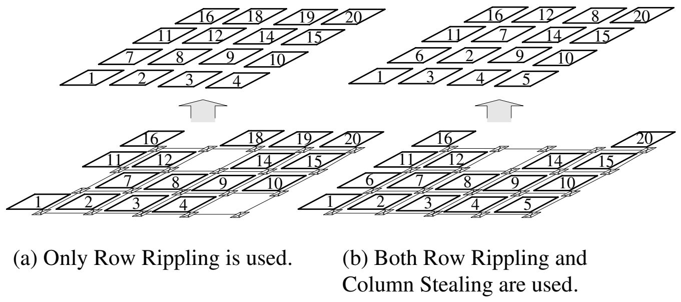
Distribute these coordinates to $(N + M - D)$ fault-free cores to construct a logical topology $T_{logical}$, in which nodes with coordinates (*i*+1, *j*), (*i*-1, *j*), (*i*, *j*+1) and (*i*, *j*-1) are four logical neighbors of node (*i*, *j*), and nodes without being assigned coordinates are left unused, satisfying:

UM of $T_{logical}$ is minimized.

Two example logical topologies for a given physical topology are shown in Fig 4. The values of *DF* and *CF* for these two logical topologies are also shown in the figure. Clearly, new topology reconfiguration algorithm needs to be developed to select the best candidate topology.

4. Proposed topology reconfiguration algorithm

As any two fault-free cores can be logical neighbors to each other, the solution space of TRP is huge. Therefore, efficient and effective heuristics should be introduced to solve this problem. In this paper, we mainly focus on the reconfiguration problems for mesh and torus topologies, which are the most widely used ones for homogeneous manycore processors. Other topologies (e.g., butterfly topology or fat tree topology) may require different optimization algorithms.



(a) Only Row Rippling is used.
(b) Both Row Rippling and Column Stealing are used.

Figure 5. Examples of *RRCS* procedure.

Simulated Annealing (SA) is a widely-used technique for combinatorial optimization problems and it is utilized to solve TRP in this work. To apply SA to our TRP problem, we first present an efficient and effective heuristic, namely Row Rippling Column Stealing (*RRCS*) algorithm, to generate a good initial solution, and then apply SA on top of it to further refine the solution. We call this strategy *RRCS*-guided simulated Annealing (*RRCS-gSA*) technique, detailed in the following.

4.1. Row rippling column stealing

It can be easily observed that the performance degradation of a logical topology is mainly caused by the physical irregularity of the logical topology compared to the reference topology. Based on this observation, the proposed *RRCS* algorithm tries to maintain the physical regularity of the logical topologies in row and in column unit.

Suppose in mesh or torus topology, there are one column of spare cores. If a row contains only one faulty core, Row Rippling is employed to reconfigure the row, in which a faulty core is replaced with its neighbor and the logical position of the core used to replace the faulty one is transferred to the next neighboring core. This process continues until the spare one is used to replace the last element in the row. An example of using row rippling in a “16+4” processor with 4×4 mesh reference topology and one column redundancy is depicted in Fig 5(a). The achieved logical topology is shown above the physical topology.

When a row contains more than one faulty cores, the rightmost faulty core is replaced using rippling. The other faulty elements within the row, however, are replaced with the elements immediately beneath them. In other words, we “steal” a fault-free core from another row within the same column. This stolen core should be considered faulty when the row containing it is reconfigured. As another example shown in Fig 5(b), to configure the uppermost row, which contains 3 faulty cores, we steal the 12^{th} and the 8^{th} fault-free cores for the left two fault cores; while the rightmost one is rippling to the 20^{th} core, as shown in the logical topology above.

In the above discussion, we provide a column of redundant cores as an example. In practice, the number of redundant cores, i.e., M, for an N-core processor should be carefully decided by the designers in advance, and may be different from the column size. This however does not affect the working mechanism of the proposed *RRCS* algorithm as it only needs to compare the number of faulty cores N_f and spare cores on each row. We are able to generate an effective logical topology as long as the number of faulty cores is less than M.

4.2. *RRCS*-guided simulated annealing algorithm

RRCS is very efficient to generate an effective initial solution, but it does not directly consider *DF* or *CF* metrics during the optimization process. We apply SA technique on top of its result to further refine the solution. During the SA optimization process, we use the UM defined in Eq. 4 as the minimization objective. The procedure of *RRCS-gSA* algorithm is described as below.

Procedure for *RRCS*-guided Simulated Annealing

```
STEP1: Use RRCS to generate an initial solution $lt_{current}$;
       $T \leftarrow TMAX$; $iter \leftarrow 0$;
STEP2: Select a $lt_{next}$ from the neighbor-set of $lt_{current}$;
       IF $UM(lt_{next}) < UM(lt_{current})$;
       THEN $lt_{current} \leftarrow lt_{next}$;
       ELSE $lt_{current} \leftarrow lt_{next}$ with probability $e^{\frac{-\Delta UM}{T}}$;
       REPEAT STEP2 $k_T$ times
STEP3: $T \leftarrow r \times T$
       IF $T \geq TMIN$ THEN $iter$++; return to STEP2;
       ELSE exit.
```

In *RRCS-gSA*, for a solution point, its neighborhood solution is generated by exchanging the coordinates of each fault-free node-pair. For example, if the current solution is

$$\begin{bmatrix} (1,0) & Faulty & Unused \\ (0,0) & (0,1) & (1,1) \end{bmatrix},$$

one of its neighbors by exchanging (1,1) and 'unused' is

$$\begin{bmatrix} (1,0) & Faulty & (1,1) \\ (0,0) & (0,1) & Unused \end{bmatrix}.$$

As can be observed from the algorithm, we evaluate k_T neighbor solutions of $lt_{current}$ at each temperature T. There are some other important parameters that determine the performance of *RRCS-gSA* algorithm, in which *r*, *TMAX* and *TMIN* are the cooling rate, the initial temperature and the solidification temperature, respectively. We can trade-off the quality of the obtained solution with computational time in the *RRCS-gSA* algorithm using the above parameters.

5. Experimental results

5.1. Experimental setup

In our simulation experiments, a synthetic traffic model is adopted, which is represented as an array of communication distribution probability [12]. For example, $[P_{1hop}, P_{2hop}, P_{3hop}, P_{>3hop}]$ = [40, 20, 20, 20] means that the probability for 1-hop communication is 40%, and is 20% for 2-hop communication, etc. We can get various traffic patterns by adjusting the probability distribution. It is important to point out that the traffic patterns are applied to logical topologies, not to physical topologies. A NoC simulation engine is implemented to incorporate the above traffic model. We obtain statistics of End-To-End delay (ETE delay), link occupation and throughput, and use them to compare the performance of different logical topologies.

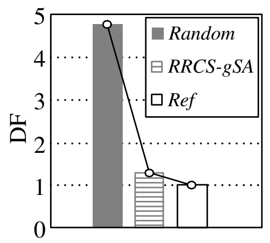
Finally, we introduce a *Random* algorithm as the baseline solution to compare with our proposed algorithm. For a given physical topology, this *Random* algorithm generates 2000 different logical topologies randomly and provides the best one as its output.

5.2. Experiment I

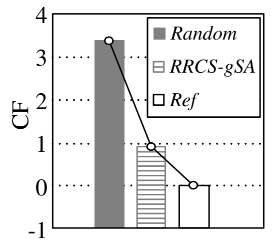
The objective of this experiment is to not only show the effectiveness of the proposed *RRCS-gSA* algorithm, but also show the effectiveness of the two evaluation metrics used in our algorithms, i.e., *DF* and *CF*.

The experimental manycore processor has a 8×8 mesh reference topology with one column spare cores and 8 randomly distributed faulty cores. We construct various physical topologies and import the logical topologies generated from *Random* and *RRCS-gSA* algorithms into the simulation environment. The obtained *DF* and *CF* values are averaged and shown in Fig 6.

First of all, it is clear that *RRCS-gSA* algorithm achieves great improvement over *Random* algorithm in terms of both *DF* and *CF*, which proves the effectiveness of the proposed algorithm.



(a) *DF* comparison.

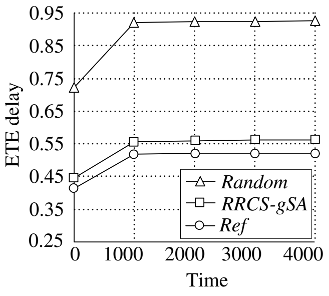


(b) *CF* comparison.

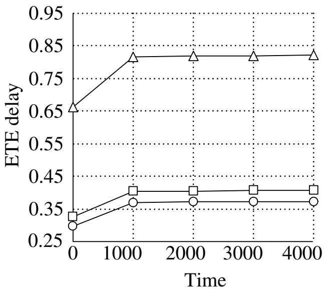
Figure 6. *DF* and *CF* comparison between *Random*, *RRCS-gSA* (w_{DF}=w_{CF}=0.5) and *Ref*

Next, we show how these two proposed evaluation metrics, i.e., *DF* and *CF*, reflect the real performance of different logical topologies. Two traffic patterns are applied: One with one-hop communication only ([100, 0, 0, 0]) while the other one with a more uniform pattern ([40, 20, 20, 20]).

ETE delay is defined as the average latency for a packet traveling from its source to its destination under certain traffic pattern. Generally speaking, shorter average distance between nodes results in lower ETE delay under the same traffic pattern. The ETE delay results are depicted in Fig 7. *Ref* represents the ETE delay of the reference topology. We can observe that logical topologies achieved by *RRCS-gSA* (w_{DF}=0.9) have much lower ETE delays than the ones obtained using *Random* algorithm, and they are also quite close to the ETE delay in *Ref* under both traffic patterns.

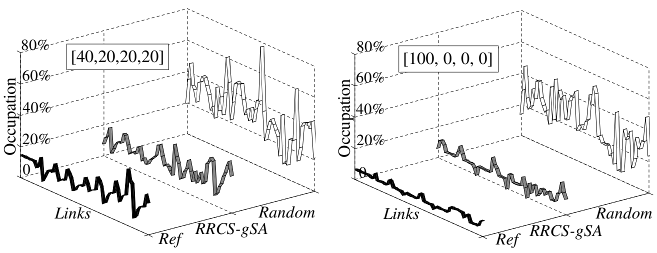


(a) traffic pattern [40, 20, 20, 20].



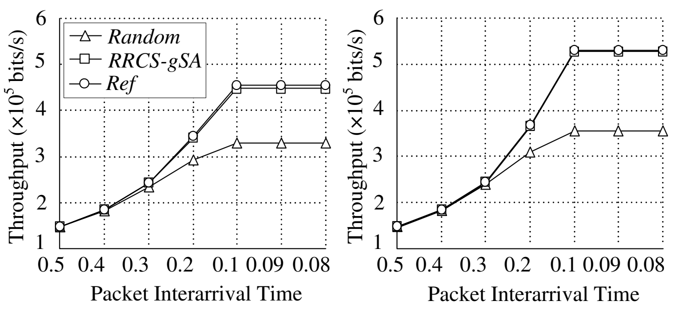
(b) traffic pattern [100, 0, 0, 0].

Figure 7. ETE delay comparison between *Random*, *RRCS-gSA* (w_{DF}=0.9, w_{CF}=0.1) and *Ref*.



(a) traffic pattern [40, 20, 20, 20]. (b) traffic pattern [100, 0, 0, 0].

Figure 8. Flow distribution comparison between *Random*, *RRCS-gSA* (w_{DF}=0.1, w_{CF}=0.9) and *Ref*.



(a) traffic pattern [40, 20, 20, 20]. (b) trafficpattern [100, 0, 0, 0].

Figure 9. Throughput comparison between *Random*, *RRCS-gSA* (w_{DF}=w_{CF}=0.5) and *Ref*.

Link occupation is defined as the percentage that the link bandwidth is occupied. Higher occupation value means more traffic traveling through the link, or in other words, this link is more congested. The average occupations of all links under different traffic patterns are depicted in Fig 8. Again, *Ref* represents the value of the reference topology.

For *Random* algorithm, the link occupation distribution is rather uneven. Some links are quite congested while the others are barely used as shown in Fig 8. Traffic distribution of *RRCS-gSA* (w_{CF}=0.9) is close to the reference topology under traffic pattern [40, 20, 20, 20] as shown in Fig 8(a), while they are more uneven when compared to the reference value under pattern [100, 0, 0, 0] as shown in Fig 8(b). It can be concluded that the performance degradation of logical topologies has greater impact on more localized traffic pattern.

Finally, we compare the throughputs of various logical topologies achieved by *Random* and *RRCS-gSA* algorithms under different traffic patterns. The average results are shown in Fig 9. As the packet inter-arrival time decreases, the NoC links tend to be saturated and we can obtain the throughput of the logical topology. From these figures, we can observe that the throughput for the logical topology obtained with *Random* algorithm is quite small. For *RRCS-gSA* (w_{DF}=w_{CF}=0.5), the obtained throughput is slightly smaller than the one in *Ref* under traffic pattern [40, 20, 20, 20], while they are almost the same as the reference value under traffic pattern [100, 0, 0, 0].

5.3. Experiment II

In this experiment, we evaluate the impact of the number of spare cores on topology reconfiguration. A 8×8 2-D mesh reference topology with 2 randomly distributed faulty cores is used. We vary the number of spare cores from 2 to 8 (i.e., S2, S4, S6 and S8 in Fig 10).

For a fixed number of faulty cores, more spare cores result in more unused cores and links according to AMAD scheme. The average distance among logical nodes and the flow distribution becomes much worse when using *Random* algorithm as can be seen in Fig 10. However, *RRCS-gSA* algorithm is very effective to avoid this situation. As can be seen in Fig 10, for *RRCS-gSA*, *DF* is slightly improved while *CF* remains nearly unchanged. Therefore, we can conclude employing more-than-necessary number of spare cores does not facilitate to boost the manycore processors' performance much after reconfiguration.

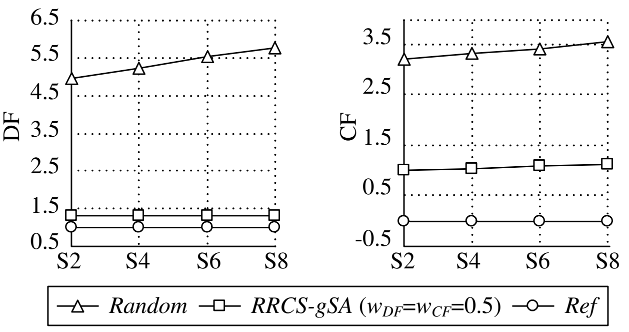


Figure 10. The impact of different number of spare cores.

6. Conclusion

In this paper, we propose to employ core-level redundancy with AMAD scheme for defect tolerance in homogeneous manycore processors. As defective cores change the physical topology of the target design, the concept of logical topology is introduced to provide a unified topology to the OS and programmers to ease their optimization. We also present an effective and efficient heuristic, namely *RRCS-gSA* algorithm, to solve the topology reconfiguration problem. Experimental results show that the proposed technique is able to dramatically boost the performance of the manycore processor when compared to a random baseline solution.

References

[1] S. Borkar. Thousand core chips - a technology perspective. *Proc. DAC*, pp. 746–749, 2007.

[2] A. Agarwal, M. Levy. The kill rule for multicore. *Proc. DAC*, pp. 750–753, 2007.

[3] Intel White Paper "From a few cores to many: a tera-scale computing research overview." *http://www.intel.com/research/platform/terascale/*.

[4] W. J. Dally, B. Towles. Route packets, not wires: on-chip interconnection networks. *Proc. DAC*, pp. 18–22, 2001.

[5] I. Koren, D. K. Pradhan. Yield and performance enhancement through redundancy in VLSI and WSI multiprocessor systems. *Proceedings of the IEEE*, 74(5):699–711, May 1986.

[6] I. Koren, Z. Koren. Defect tolerance in VLSI circuits: techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819-1838, Sep.1998.

[7] S. Premkishore, S. W. Keckler, C. R. Moore, D. Burger. Exploiting microarchitectural redundancy for defect tolerance. *Proc. ICCD*, pp. 481–488, 2003.

[8] E. Schuchman, T. N. Vijaykumar. Rescue: a microarchitecture for testability and defect tolerance. *Proc. ISCA*, pp. 160–171, 2005.

[9] P. J. Tan, et al. Testing of UltraSPARC^{TM} microprocessor and its challengess. *Proc. ITC*, pp. 1–10, 2006.

[10] M. Fukushi, Y. Fukushima, S. Horiguchi. A genetic approach for the reconfiguration of degradable processor arrays. *Proc. DFT*, pp. 63–71, 2005.

[11] Application software considerations for numa-based systems. *http://www.microsoft.com/whdc/system/platform/server/datacenter/numa_isv.mspx*.

[12] Z. H. Lu, A. Jantsch. Traffic configuration for evaluating networks on chips. *Proc. IEEE Int. Workshop System-on-Chip for Real-Time Applications*, pp. 535–540, 2005.

[13] W. Stallings. *Operating systems: internals and design principles*. Prentice Hall, 2000.