A Greedy Approach to Tolerate Defect Cores for Multimedia Applications

Ke Yue*, Soumia Ghalim*, Zheng Li*, Frank Lockom*, Shangping Ren*, Lei Zhang†, and Xiaowei Li†
*Department of CS
Illinois Institute of Technology
Chicago, Illinois 60616
Email:{kyue, sghalim, flockom, zli80, ren}@iit.edu
†Key Laboratory of CSA
Institute of Computing Technology
Chinese Academy of Sciences
Email: {zlei, lxw}@ict.ac.cn

***Abstract*—Computation-intensive multimedia applications are emerging on mobile devices.System-on-Chip (SoC) offers high performance at a decreased size for these devices. SoC often integrates tens of cores and uses Network-on-Chip (NoC) as its communication infrastructure. To ensure high yield of manycore processors, core-level redundancy is often used as an effective approach to improve the reliability of manycore chips. However, when defective cores are replaced by redundant ones, the NoC topology changes. As a result, a fine-tuned application based on timing parameters given by one topology may not meet the expected timing behavior under the new one. To address this issue, we first define a metric that can measure the timing resemblance between different NoC topologies. Based on this metric, we develop a greedy algorithm to reconfigure a defect-tolerant manycore platform and form a unified application specific virtual topology on which the timing variations caused by the reconfiguration are minimized. Our simulation results clearly indicate the effectiveness of the developed algorithm.**

I. Introduction

As technology advances, manycore architectures are becoming mainstreams for a large spectrum of applications, including real-time multimedia applications. As there are many cores on-chip, such architectures typically employ the scalable Network-on-Chip as the communication backbone among processing cores.

Many challenges need to be tackled for the design of NoC-based manycore processors. Permanent core failure due to manufacturing defects or transistors wear-out is one of the most challenging problems. According to Sperling's report [1], for a Cell processor, without considering defect tolerance during the architecture design phase, even under the best case, the yield can be as low as 10 to 20 percent.

As there are many light weight cores on-chip, and each core occupies a small area of the chip footprint, core-level redundancy is proven to be an efficient technique to tackle the above core failure problem [2]. If N cores are expected to be provided to customers, M redundant cores will be added on chip. Defective cores can be replaced with redundant ones, thus providing the demanded computing capability. However, when a defective core is replaced by a redundant core, it is possible that the on chip topology, i.e., the interconnect relationship among cores is changed, for example from a regular 2D mesh topology to an irregular topology. Such an underlying topology with possible defective cores is called a physical topology. Different chips may have different physical topologies with different failure bitmaps.

It would be a great burden for application developers to consider these topologies upon which their programs are designed, deployed and optimized. Therefore, topology virtualization is proposed to isolate various underlying physical structures, and provide programmers with a unified interface [3].

Prior research on manycore topology virtualization mainly focused on general computing domain and the methods proposed intend to achieve better performance, in terms of communication latency and network throughput [2], [4], [3]. However, the goals differ from the above for applications with timing requirements, such as multimedia applications. For multimedia applications, rather than performance, the most important property is the synchronization between different medias, such as the synchronization between video and audio streams, between user inputs and device outputs. Therefore, timing similarity instead of high performance is preferred to avoid introducing extra cost in redesign, re-implementing, and retesting the system when defective cores are replaced by redundant ones.

Based on the above, in this paper we revisit topology virtualization techniques in the real time domain. Figure 1 depicts the framework. By virtualizing the cores in a physical topology, we can build a virtual topology that is isomorphic to the reference design. In particular, when an application is mapped to a manycore platform, i.e., tasks are fixed to virtual cores, if cores on which application tasks are deployed become defective, we virtualize redundant cores to replace them. The task is deployed on the same virtual core from the application's view but on a different physical core from hardware's perspective.

In this paper, we focus on homogeneous manycores, in

The research is supported in part by NSF CNS 1018731, NSF Career 0746643, and NSF CNS 1035894.

which all cores have the same performance, such as operating frequencies. Therefore, deploying tasks on different cores only impacts the communication time among tasks. For example, the physical distance between two communicating cores may be changed. These changes may impact the application's timing behavior and cause timing synchronization violations. Therefore, it is important to maintain applications' timing similarity as much as possible before and after topology virtualization.

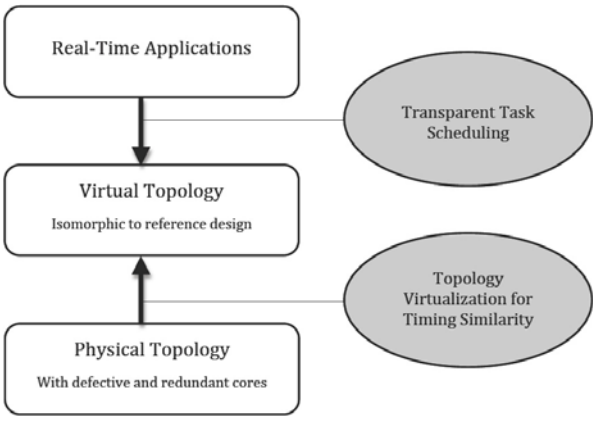


Fig. 1: Topology virtualization for application's timing similarity

The contribution of this paper is twofold. First we propose a metric to evaluate the timing similarity of isomorphic virtual topologies with different underlying physical topologies. Second we propose an efficient heuristic algorithm (polynomial time) to solve the defective core replacement problem.

The rest of the paper is structured as follows. Prior research is discussed in Section II. We motivate and formulate the problem of finding an appropriate virtual topology in Section III. A heuristic algorithm, i.e. greedy algorithm, is presented in Section IV and its effectiveness is shown through experiments in Section V. Finally, we conclude in Section VI.

II. Related Work

As mobile computing becomes pervasive, manycore NoC platforms have been used to accommodate multimedia applications. Much of the research has been done in the area of NoC topology explorations. For instance, Lankes et. al [5] studied a NoC topology exploration based on a real-word mobile multimedia application example, by using an abstract simulation model, they pointed out that the enhanced unidirectional ring topology has the best performance with latency and chip area taken into consideration. Ma et. al [6] conducted system-level exploration of mesh-based NoC architectures for multimedia applications. This exploration contains two correlated questions, i.e., given an application and node processing throughput, how to determine the optimal size for the mesh and the link bandwidth; and conversely, given an application and its mapping to a given-sized mesh, how to optimally size the node processing throughput and link bandwidth. They established a simulation platform and proposed an exploration approach to obtain the optimal design for specific applications. Lee et.al [7] focused on designing space exploration for on-chip multimedia applications, by comparing and contrasting the P2P and NoC-based implementations of a real multimedia application, they concluded that NoC design scales much better in terms of area, performance, power and overall design effort.

Optimization and reconfigurations for multimedia applications on NoC platforms have also been studied extensively. Hasson [8] developed a model that enables partial reconfiguration of NoCs and proposed a mapping algorithm that uses the model to map multiple applications onto a NoC with undisrupted quality-of-service during the reconfiguration. In order to design the optimal application-specific NoC platforms, Papadopoulos et.al [9] presented a new tool based on the Nostrum NoC simulator to provide an essential evaluation metric required for achieving the optimization in dynamic network and multimedia applications at the system level. Mostafavi et.al [10] devised an iterative rate control algorithm to achieve the optimal solution to rate control problem for applications with scalable multimedia services in NoC architectures. Haiyun [11] addressed a mapping algorithm of irregular mesh NoC for portable multimedia applications to achieve the minimum communication cost with certain constrains.

Virtualization provides a unified hardware interface for applications. The topology virtualization problem for *general purpose computing* is discussed thoroughly in [2], [4], [3]. Performance degradation of virtual topologies when compared to the topology initially designed, i.e., the reference topology, is evaluated by using two metrics, i.e., distance factor (DF) and congestion factor (CF). The DF is the average hop count between a core and all of its virtual neighbors, which determines the zero-load communication latency of a virtual topology. CF is used to evaluate the channel load distribution among links under certain routing algorithm (e.g., XY-routing). As the more balanced the channel load, the closer the throughput of the network is to the reference one, a reconfigured topology that balances traffic more evenly across all NoC links is preferred. The topology virtualization problem is then formalized based on these two performance metrics. A heuristic called Row Rippling Column Stealing (RRCS) is proposed in [2]. The essence of the heuristic is to maintain the physical regularity of reconfigured virtual topologies in both row and column units, and hence to maximize performance. It is also worth noting that the RRCS method is application agnostic, where as the application is the primary consideration in our greedy method. Our comparison shows that when the application is known e.g. mobile/embedded application, this information can be used to achieve better timing similarity.

Rather than using performance as a single objective, manycore topology virtualization for applications with specific timing requirements, such as real-time applications, also needs to consider maintaining timing similarities among different topologies. Many notions of timing behavior similarities have

been proposed in various literature. For instance, Huang *et al*. [12] investigate the real-time property preservation between similar timed state sequences (execution traces of timed systems). The authors later extend the results to concurrent real-time systems in [13]. Henzinger *et al*. [14] define quantitative notions of timed similarity and bisimilarity on timed systems. They also give algorithms to compute the similarity distance between two timed systems modeled as timed automaton. Recent research results show that the timing behavior of a system can be characterized by a feasible region defined by the system's timing constraint set [15], [16].

Different from prior work listed above, we focus on topology virtualization for applications in which rather than performance, timing similarity is the paramount requirement. The virtualization objective is to use redundant cores provided on the chip to replace defective cores and at the same time to maximize the timing resemblance of the newly configured one to the manycore topology that is initially designed. The reconfiguration procedure is intended for offline implementation in order to improve the yield of multicore NoCs.

III. Motivation and Problem Formulation

In this section, we first present the motivation of this work, and then formulate the problem the paper is to address.

A. Motivation

Assume a multi-media application is to be deployed on a 9 core processor with 3×3 2D mesh topology. The 3×3 mesh is referred as *reference topology*. To enhance the yield of the 9-core chip, we provide another 3 redundant cores as shown in Figure 2. If, for instance, core C_4 is defective, the remaining defect-free cores form a new *physical topology*. We can virtualize the defect-free cores and provide applications a virtual 3×3 2D mesh topology. There are several ways to do this, for instance, in Figure 2, the three redundant cores, i.e., R_0, R_1, or R_2, can be used to replace the defect core C_4, and hence generate three different virtual topologies. We use i and (i) to index a physical core and a virtual core, respectively. Even though applications are given a unified 3×3 2D mesh, but different virtual topologies have different properties, such as latency and throughput.

For instance, consider the filter application given in [17]. The application consists of six communicating tasks as shown in the application layer of Figure 3, where the value on each edge represents data transfer rate between two communicating tasks. If the six communicating tasks are mapped to a 3×3 mesh topology as shown in Figure 3, i.e., task v_1 and v_4 are mapped to $C_{(3)}$, v_2 and v_3 to $C_{(1)}$, and v_5 and v_6 to $C_{(4)}$, then the traffic flow among two virtual cores are the summation of data transfer rates that go through the two cores. For the given mapping, the communication flow between virtual core $C_{(1)}$ and $C_{(3)}$ is 300 bits/cycle which is the summation of data transfer rate between v_1 and v_2, and between v_4 and v_3. Figure 4 gives the traffic flow among different cores.

More generally, for a given application that is mapped to s cores supported by a $n \times (n+1)$ physical topology with n redundant cores, if among the s cores, p ($p \leq \min\{s, n\}$) cores become defective, we have $\binom{n}{p}p!$ number of different ways to form a virtual topology which is topologically isomorphic to the application's initial design. Then, the question is which one we should choose.

Consider again the filter application and its initial mapping (Figure 3). When there is no defect cores, the resultant traffic flow between virtual cores is listed in Table I. The traffic rate in the table indicates the number of bits transferred between two virtual cores per cycle and the hop count is the number of physical hops between two virtual cores using XY-routing algorithm. Using the NIRGAM NoC interconnect routine and application modeling tool [18], we obtain that the average packet latency for this initial reference topology is 26.34 cycles.

However, if the physical core C_4 becomes defective, it can be replaced by one of the redundant cores R_0, R_1, or R_2. Depending on which redundant physical core is used, the resulting topologies' hop count, traffic rate, and average package latency under XY routing between virtual cores are different as shown in Table II. From the table, it is clear that for the specific application, using R_0 as the new virtual core $C_{(4)}$

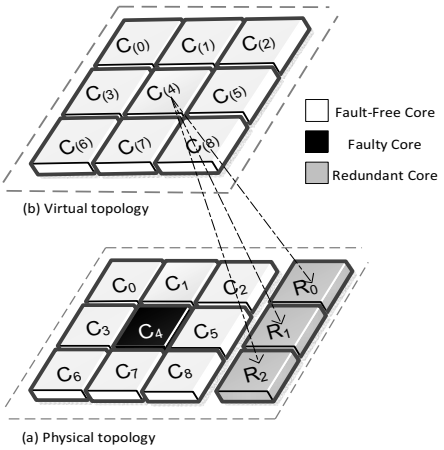


Fig. 2: Physical topology and virtual topology

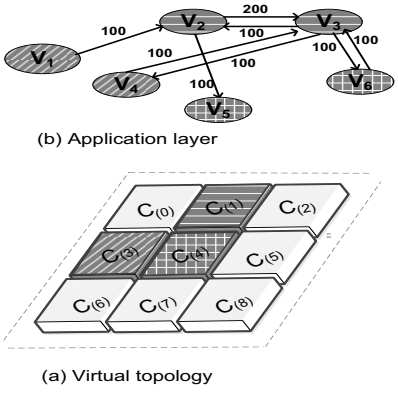


Fig. 3: Mapping application to virtual topology

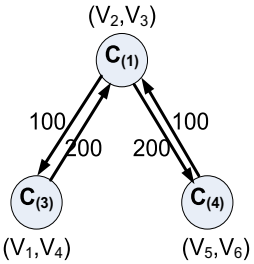


Fig. 4: Traffic flow between virtual cores

has the least average packet latency difference from the initial reference topology which has average latency of 26.34 cycles. Hence, R_0 is the best choice if *similar* timing behavior is used as the evaluation metric. In fact, latency is the most important metric to evaluate interconnection network. If two networks have the same timing behavior, i.e., the same task-to-task communication latency, we can safely assume that applications have the same timing properties under homogeneous manycore platforms.

Traffic Between Virtual Cores	Corresponding Physical Cores	Traffic Rate	Hop Count
$C_{(1)} \leftrightarrow C_{(4)}$	$C_1 \leftrightarrow C_4$	300	1
$C_{(1)} \leftrightarrow C_{(3)}$	$C_1 \leftrightarrow C_3$	300	2
$C_{(3)} \leftrightarrow C_{(4)}$	$C_3 \leftrightarrow C_4$	0	1

TABLE I: Scheduling on initial defect-free topology

Traffic Between Virtual Cores	Corresponding Physical Cores	Traffic Rate	Hop Count	AvgPack Latency
$C_{(1)} \leftrightarrow C_{(4)}$	$C_1 \leftrightarrow R_0$	300	2	26.73
$C_{(1)} \leftrightarrow C_{(4)}$	$C_1 \leftrightarrow R_1$	300	3	27.45
$C_{(1)} \leftrightarrow C_{(4)}$	$C_1 \leftrightarrow R_2$	300	4	28.9

TABLE II: Virtualizing different redundant cores for $C_{(4)}$

B. Problem Formulation

Before formulating the problem of finding an appropriate virtual topology for a given application, we make two assumptions regarding the NoC-based manycore system. First, the manycore system is a 2-D mesh network running under XY routing. Second, defective cores can only be replaced by redundant ones so that the defect-free cores responsible for other applications are not disturbed.

In order to analyze the timing resemblance between the reference and the virtual topologies, the timing behavior of a NoC-based manycore system upon which a specific application is deployed has to be first defined. As for homogeneous manycore systems, processors speeds are the same. Hence, on-chip communication becomes the dominant factor that differentiates various virtual topologies' timing behaviors.

We use *traffic flow occupancy* $F_{(i),(j)}$ to quantify the communication pattern from virtual cores $C_{(i)}$ to $C_{(j)}$. The formal definition is given below:

Definition 1 (Traffic Flow Occupancy): For a given virtual topology T^k, the traffic flow occupancy from virtual core $C_{(i)}$ to core $C_{(j)}$ is defined as

$$F^k_{(i),(j)} = \begin{cases} \tau_{(i),(j)} \times H^k_{(i),(j)} & \text{if the application uses } C_{(i)}, C_{(j)} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\tau_{(i),(j)}$ is the communication rate from virtual core $C_{(i)}$ to virtual core $C_{(j)}$, and $H^k_{(i),(j)}$ is the number of hops from $C_{(i)}$ to $C_{(j)}$ under a given topology and routing algorithm. In this paper a 2D mesh topology with XY routing is used.

□

For instance, according to Table I, $\tau_{(1),(3)}$ and $H^k_{(1),(3)}$ are 300 and 2, respectively, then the traffic flow occupancy $F^k_{(1),(2)}$ from virtual core $C^k_{(1)}$ to $C^k_{(3)}$ is equal to 600.

For two different virtual topologies, T^k and T^r, their traffic flow occupancy difference from virtual core $C_{(i)}$ to $C_{(j)}$ can be calculated by

$$\Delta^{k,r}_{(i),(j)} = |F^k_{(i),(j)} - F^r_{(i),(j)}| \tag{2}$$

where $F^k_{(i),(j)}$ and $F^r_{(i),(j)}$ are application's traffic flow occupancy from virtual cores $C_{(i)}$ to $C_{(j)}$ for virtual topology T^k and T^r, respectively. Clearly, the smaller the occupancy differences among *all* virtual core pairs, the higher the timing similarity among the two topologies. A metric which is similar to the average and standard deviation is used to characterize the change of the traffic flow occupancy. We use Ave and Var to model the traffic flow occupancy difference of the entire topology and the individual communication links respectively. Their definitions are given below:

Definition 2: (*Normalized Average Traffic Flow Occupancy Difference*)

For a given topology T^k which is topologically isomorphic to the reference topology T^r of size $n \times n$, the normalized average traffic flow occupancy difference is given by

$$Ave^k_r = \frac{\sum_{(i,j)\in F} \Delta^{k,r}_{(i),(j)}}{\Psi_r |F|} \tag{3}$$

where F is the set of traffic flows of the application and Ψ_r is the average traffic flow occupancy of the reference topology T^r given by

$$F = \{(i,j) | F^r_{(i),(j)} \neq 0, 0 \leq i,j < n^2\} \tag{4}$$

$$\Psi_r = \frac{\sum_{(i,j)\in F} F^r_{(i),(j)}}{|F|} \tag{5}$$

□

Definition 3: (*Normalized Variation of Traffic Flow Occupancy Difference*)

For a given topology T^k which is topologically isomorphic to the reference topology T^r of size $n \times n$, the normalized variation of traffic flow occupancy difference is given by

$$Var_r^k = \sqrt{\frac{\sum_{(i,j)\in F} (\frac{\Delta^{k,r}_{(i),(j)}}{\Psi_r} - Ave_r^k)^2}{|F|}} \quad (6)$$

□

Definition 4: (*Virtual Topology Timing Similarity*)
For a given topology T^k which is topologically isomorphic to the reference topology T^r of size $n \times n$, their timing similarity $\chi(T_r^k)$ is defined by

$$\chi(T_r^k) = w_a \times Ave_r^k + w_v \times Var_r^k \quad (7)$$

where w_a and w_v ($w_a + w_v = 1$) are the weights applied to the Ave and Var respectively. Note that throughout this paper we let $w_a = w_v = 0.5$.

□

With the similarity metric, we formulate the problem that the paper is to address as follows:

Problem 1: For a given reference topology T^r supported by an $n \times (n+1)$ physical topology with n redundant cores, and on which a given application $\mathcal{A}$ is deployed, if cores used by the application $\mathcal{A}$ become defective, construct a virtual topology T^{opt} for the given application so that it is topologically isomorphic to T^r but without defective cores, and further it satisfies the requirement (8)

$$\chi(T_r^{opt}) = \min\{\chi(T_r^k) | T^k \ is \ isomorphic \ to \ T^r\} \quad (8)$$

□

As defect-free cores in a physical topology can be placed in any location in a virtual topology, the solution space of the topology reconfiguration problem, i.e., all possible virtual topologies topologically isomorphic to initially designed reference topology, is combinatorially large. Therefore, efficient heuristics are needed in order to solve the problem.

IV. Reconfiguration Algorithms

In this section, we present a heuristic reconfiguration algorithm, the greedy algorithm.

A. Greedy Algorithm

The key concept of a greedy algorithm is to find a local optimal solution and combine them to get a global solution. Based on this strategy, Consider the application $\mathcal{A}$ which is mapped to a reference topology T^r. T^r is supported by an $n \times (n+1)$ physical topology with n redundant cores. The objective then is to replace p, $0 \le p \le n$ defective cores.An approximation (T^{approx}) to the optimal T^{opt} can be obtained by finding the local optimal topology T_i^{opt}, $i = 0, \cdots, p$, for the i^{th} defective core being replaced by a redundant one.

To illustrate the procedure of the greedy algorithm, again consider the 3×3 mesh topology with applications mapped to cores as shown in Figure 2. Assume the physical cores C_3 and C_4 become defective, we need to choose 2 redundant cores to form a new virtual topology that is the most similar to the initial reference topology from timing perspective.

First, we need to compute the overall traffic flow occupancy for each defective core to determine the order in which the defective cores are replaced. For a specified virtual core $C_{(m)}$ in virtual topology T^k, the total traffic flow occupancy is defined as:

$$TC^k_{(m)} = \sum_{0\le i\le n^2} F^k_{(m),(i)} + \sum_{0\le i\le n^2} F^k_{(i),(m)} \quad (9)$$

The virtual core with the largest traffic flow occupancy should be replaced first. For the given example, $TC^k_{(3)} = 600$ and $TC^k_{(4)} = 300$, thus the order for replacing the defective cores should be C_3 and then C_4.

Next, replace the defective core C_3 with three redundant cores R_0, R_1 and R_2 to get three respective virtual topologies T_r^0, T_r^1, and T_r^2. By (7), we obtain the corresponding $\chi(T_r^0), \chi(T_r^1)$ and $\chi(T_r^2)$ which are 0, 4.69 and 9.38, respectively. As $\chi(T_r^0)$ has the smallest value, i.e., T_r^0 most resembles T^r, therefore T_r^0 is the local optimal virtual topology T_1^{opt} when the first defective core C_3 is replaced. We then use R_0 to replace core C_3 which forms a new virtual core $C_{(3)}$ to physical map.

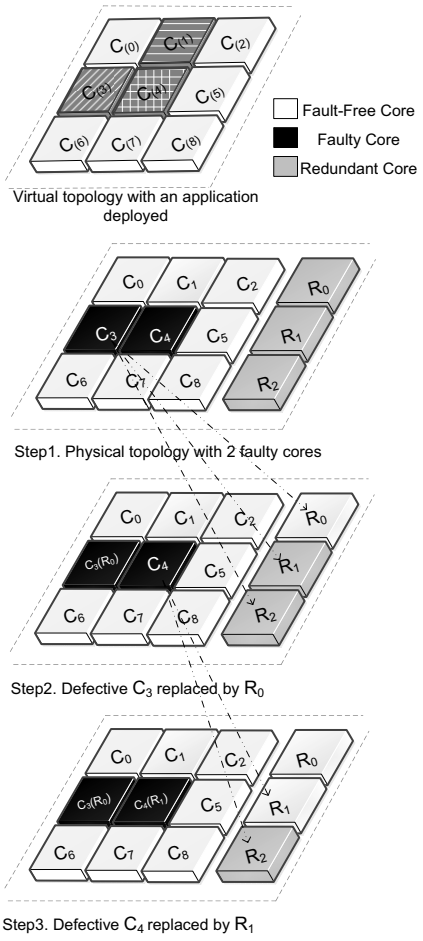


Fig. 5: Examples of greedy algorithm.

Lastly, replace the defective core C_4 with one of two remaining redundant cores R_1 and R_2, we obtain two possible virtual topologies T_r^1, T_r^2. The corresponding $\chi(T_r^1)$ and $\chi(T_r^2)$ is 6.93 and 10.39, respectively. Therefore, T_2^{opt} is T_r^1 and R_1 is used to replace C_4. The aproximation to the optimal virtual topology T_r^{approx} includes physical cores C_1, R_0, and R_1, as shown in the Figure 5. The virtual cores $C_{(3)}$ and $C_{(4)}$ are re-mapped to physical cores R_0 and R_1.

Figure 5 depicts the process where every time a redundant core is selected to replace a defective one, its color is changed from gray to white to indicate that it becomes part of the topology.

Algorithm 1 summarizes the above steps.

Algorithm 1 `Greedy algorithm` $(T^r, \mathcal{A}, p, n)$

1: **Sort** the order for replacing the defective cores based on (9)
2: assume all the defective cores are removed from the reference topology
3: **for** $i = 1, \cdots, p$ **do**
4: add C_i to the reference topology;
5: replace the defective core C_i by the redundant core j that has the smallest $\chi(T_i^j)$ and obtain the local optimal topology T_i^{opt}
6: Remove redundant core R_j from the redundant core set;
7: Use T_i^{opt} as the new reference topology;
8: **end for**
9: **return** T_p^{opt}

It is not difficulty to see that the time complexity for the greedy algorithm is $O(n^2)$.

V. Experimental Results

In this section, we use the NIRGAM (NoC Interconnect Routing and Application Modeling) [18] simulator to compare the performance among different reconfiguration algorithms. In particular, we first compare the average latency using the greedy algorithm and RRCS algorithm [2]. We then compare the two algorithms via the task-to-task traffic latency. The result shows that our greedy algorithm reaches a better timing similarity result than RRCS.

A. Experiment Setup

NIRGAM is a modular and cycle accurate simulator developed in SystemC. In NIRGAM, a 2D NoC mesh of tiles can be simulated by different design options, e.g., virtual channels, clock frequency, buffer parameters, routing mechanisms and applications patterns, etc. Each NIRGAM tile consists of various components, such as input channel controller, virtual channel allocator, output channel controller, and IPcores. Each IPcore is attached to a router/switch by means of a bidirectional core channel.

In our experiment, each tile is a traffic generator that keeps a constant data transfer rate for a specified communication graph. We use wormhole switching and deterministic XY routing on the NoC. Each packet contains one flit of 8 bytes and the buffer size for the FIFO input chancel is 2 flits. Each physical link has 4 virtual channels. All tasks are executed concurrently and begin to send packets after a 5 cycle warmup time. Tasks continue to send packets for an additional 3000 cycles after the warmup time.

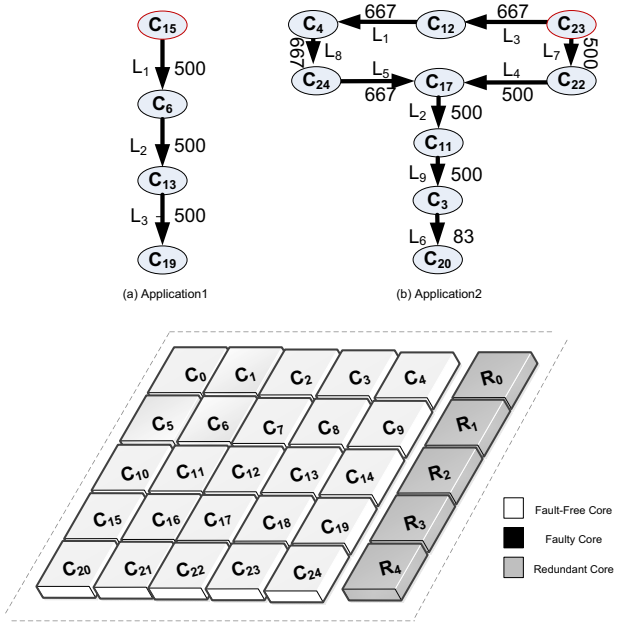
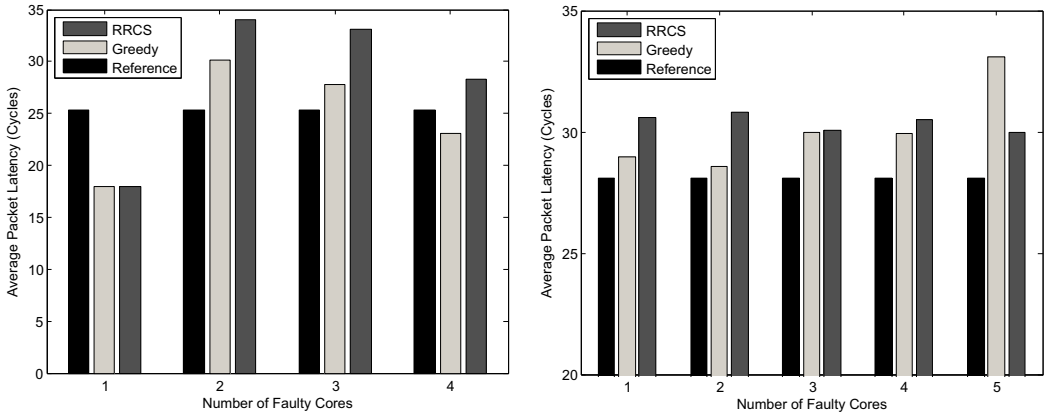


Fig. 6: The application's task graphs and the mapping generated to a 5×5 mesh

B. Comparison with RRCS

In this experiment, we evaluate the effectiveness of the greedy algorithm using task graphs provided by the embedded system benchmark suite (E3S)[19]. We choose two applications from the Office Automation Benchmark provided by the suite, i.e., *application1* and *application2*, whose tasks perform dithering, image rotation and text processing, and obtain the communication relationship among tasks as shown in Figure 6 (a) and (b), respectively. Based on [20], the tasks are mapped to a 5×5 2D mesh with additional 5 spare cores as shown in Figure 6. The numbers attached to the graph nodes represent the cores assigned to the tasks after the mapping. For instance, the nodes of *application1* are mapped to the physical cores C_{15}, C_6, C_{13} and C_{19}, respectively. The values attached to the edges, such as the value 500 between C_{15} and C_6, corresponds to the data transfer rate between the two cores in MB/s. The data transfer rate is calculated based on the packets transferred per cycle between two tasks using the task graph mentioned in [20]. Next, we assume there are 1, 2, 3 and 4 randomly distributed defective cores among the cores occupied by the two applications.

With *application1*, when there is only one defective core, it can be randomly distributed among the four cores the application uses. For each of these cases, a new virtual topology is



(a) Application 1 (b) Application 2

Fig. 7: Average latency between virtual topologies generated by greedy and RRCS algorithms(closer to reference is better)

generated using the greedy and RRCS algorithms to replace the defective core. We then calculate the average packet latency. Similarly, when there are two defective cores, there are six different ways these defective cores can be located, again, for each case, we use the three different algorithms to find a new virtual topology and calculate the average packet latency. Repeat for the cases with three and four defective cores. Figures 7a and 7b depict the results for *application1* and *application2* respectively. Overall, for both applications the greedy algorithm generates a topology whose average latency varies by no more then 18% from the reference one.

In the previous experiments, we focus on the average latency for all links. Next, we investigate the latency for each individual link among the tasks. We evaluate task-to-task traffic latency for *application2*.

As seen in Figure 6(b), *application2* has 9 traffic links among tasks. Tasks maintain the constant bit rate on each edge which is given in in Figure 6(b). The communication latency for each traffic flow L_i, $i = 1, \ldots, 9$, represents the timing interval between sending the first packet on the source core and receiving the last packet on the destination core. For example, on link L_6, the source core $C_{(3)}$ sends a packet to $C_{(20)}$ every 48 cycles to keep the traffic rate listed on the communication graph. So the last packet is sent at 2928 and the destination core receives the last packet at 2942.

If any core is defective, the irregularity of the underlying physical topology brought by the presence of defective cores causes inevitable changes in the timing behavior, and the communication latencies for some flows indeed change significantly. For instance, we assume the cores C_3 and C_4 are defective and different cores are used to replaced them based on different reconfiguration algorithms. The communication latencies for the traffic flow L_i in the reference topology and topologies generated using RRCS and greedy are shown in Figure 8.

C. Discussion of Results

Some observations are made concerning Figures 7 and 8. First it is important to restate that lower latencies are not necessarily desirable. The objective is to minimize the delta from the reference mapping, without regard to its direction. This allows fewer assumptions to be made about the objectives of the reference mapping and thus the application. It is assumed the mapping is valid and that a similar mapping will also be valid.

Looking at Figure 7 it can be seen that RRCS performs better than greedy in the case that five cores are defective. RRCS attempts to maintain the regularity of the virtual cores with respect the their initial locations instead of moving only the defective cores. This will avoid the problem seen for five cores where the right most column is highly utilized which results in congestion among those links. A potential solution is to assume the redundant cores are distributed evenly throughout the NoC. However, their placement in the right most column is an assumption made by the RRCS algorithm and hence we adopted here for better comparison. A similar explanation can be given regarding the poor similarity of Link 6 in Figure 8. Notice however that in this case greedy performs better than RRCS which brings us to our final note.

Both the greedy and RRCS approach are heuristic algorithms and may not give predictable solutions. Indeed, much work remains to fully understand the greedy algorithm and its performance under different conditions.

VI. Conclusion

There are many considerations when virtualizing manycore systems in presence of manufacturing defects and device wear-outs for multimedia applications. The desired hardware architecture, timing requirements, and the on-chip redundancy distributions must be understood. By taking these factors into consideration, the developed heuristic algorithm allows us to find the virtual topology that is most similar, in terms of their timing behaviors captured by their average pair-wise

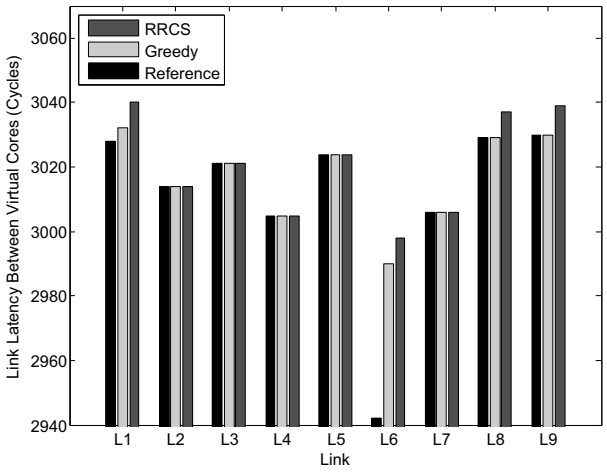


Fig. 8: Task-to-task latency comparison for *application2*.

network latencies, to the reference topology an application is initially designed on. The simulation results show that with our approach, the timing differences between the reconfigured topology and reference topology introduced by replacing defective cores are no more than 18% for the examples given.

Our immediate next step is to continue to better understand the performance of the greedy algorithm. We will evaluate the approach using many random applications across different dimensions. Specifically we will look at the performance related to the number of tasks in the application, the number of nodes in the NoC and the utilization of the links bandwidth. We will also compare with the optimal solution when the number of tasks remains small.

The research presented in the paper is only the first step toward applying virtualization technologies to real-time multimedia applications. We are all aware that for hard real-time applications, minimizing average pair-wise communication latency change may not guarantee stringent timing properties required by these applications. In the future we will study how reconfiguration may impact hard real-time applications, and investigate virtualization techniques that guarantee deadline satisfactions.

REFERENCES

[1] E. Sperling, "Turn down the heat. . . please," March 2007.

[2] L. Zhang, Y. Han, Q. Xu, and X. Li, "Defect tolerance in homogeneous manycore processors using core-level redundancy with unified topology," in *Proceedings of the conference on Design, automation and test in Europe*. ACM, 2008, pp. 891–896.

[3] L. Zhang, Y. Han, Q. Xu, X. Li, and H. Li, "On topology reconfiguration for defect-tolerant NoC-based homogeneous manycore systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 1173–1186, 2009.

[4] L. Zhang, Y. Yu, J. Dong, Y. Han, S. Ren, and X. Li, "Performance-asymmetry-aware topology virtualization for defect-tolerant NoC-based many-core processors," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 1566–1571.

[5] A. Lankes, A. Herkersdorf, and H. R. Sören Sonntag, "Noc topology exploration for mobile multimedia applications," in *Proceedings of the 16th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2009*, 2009, pp. 707 – 710.

[6] N. Ma, Z. Lu, Z. Pang, and L. Zheng, "System-level exploration of mesh-based noe architectures for multimedia applications," in *Proceedings of the 2010 IEEE International SOC Conference (SOCC)*, 2010, pp. 99 –104.

[7] H. G. Lee, U. Y. Ogras, R. Marculescu, and N. Chang, "Design space exploration and prototyping for on-chip multimedia applications," in *Proceedings of the 43rd ACM/IEEE Conference on Design Automation*, 2006, pp. 137 – 142.

[8] A. Hansson, M. Coenen, and K. Goossens, "Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, DATE '07*, 2007, pp. 1 – 6.

[9] L. Papadopoulos, S. Mamagkakis, F. Catthoor, and D. Soudris, "Application - specific noc platform design based on system level optimization," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2007, pp. 311 – 316.

[10] S. Mostafavi, A. Khonsai, M. S. Talebi, and M. Ould-Khaoua, "Rate control for scalable multimedia applications in network-on-chips," in *Proceedings of the Eighth International Conference on Scalable Computing and Communications*, 2009, pp. 621 – 625.

[11] G. Haiyun, L. Changwen, and S. Shu, "Research on mapping algorithm of irregular mesh noc for portable multimedia appliances," in *Proceedings of the IET Conference on Wireless, Mobile and Sensor Networks*, 2007, pp. 697 – 701.

[12] J. Huang, J. Voeten, and M. Geilen, "Real-time property preservation in approximations of timed systems," 2003.

[13] ——, "Real-time property preservation in concurrent real-time systems," in *Proc. of 10th RTCSA*. Citeseer, 2004.

[14] T. Henzinger, R. Majumdar, and V. Prabhu, "Quantifying similarities between timed systems," *Formal Modeling and Analysis of Timed Systems*, pp. 226–241, 2005.

[15] Y. Yu, S. Ren, and O. Frieder, "Feasibility of semiring-based timing constraints," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 4, p. 33, 2010.

[16] Y. Yu, S. Ren, and X. Hu, "A metric for judicious relaxation of timing constraints in soft real-time systems," in *15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2009, pp. 163–172.

[17] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 355–362.

[18] L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, and A. Narayanan, "NIRGAM: a simulator for NoC interconnect routing and application modeling," *Design Automation and Test in Europe (DATE), Nice, France*, 2007.

[19] R. Dick, "Embedded system synthesis benchmarks (e3s)," http://www.ece.northwestern. edu/ dickrp/e3s, 2007.

[20] A. Sharifi, H. Zhao, and M. Kandemir, "Feedback control for providing QoS in NoC based multicores," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 1384–1389.