Hungarian Algorithm Based Virtualization to Maintain Application Timing Similarity for Defect-Tolerant NoC

Ke Yue, Frank Lockom, Zheng Li, Soumia Ghalim, and Shangping Ren
Department of CS
Illinois Institute of Technology
Chicago, Illinois 60616
Email: {kyue, flockom, zli80, sghalim, ren}@iit.edu

Lei Zhang and Xiaowei Li
Key Laboratory of CSA
Institute of Computing Technology
Chinese Academy of Sciences
Email: {zlei, lxw}@ict.ac.cn

***Abstract*—Homogeneous manycore processors are emerging in broad application areas, including those with timing requirements, such as real-time and embedded applications. Typically, these processors employ Network-on-Chip (NoC) as the communication infrastructure and core-level redundancy is often used as an effective approach to improve the yield of manycore chips. For a given application's task graph and a task to core mapping strategy, the traffic pattern on the NoC is known a priori. However, when defective cores are replaced by redundant ones, the NoC topology changes. As a result, a fine-tuned program based on timing parameters given by one topology may not meet the expected timing behavior under the new one. To address this issue, a timing similarity metric is introduced to evaluate timing resemblances between different NoC topologies. Based on this metric, a Hungarian method based algorithm is developed to reconfigure a defect-tolerant manycore platform and form a unified application specific virtual core topology of which the timing variations caused by such reconfiguration are minimized. Our case studies indicate that the proposed metric is able to accurately measure the timing differences between different NoC topologies. The standard deviation between the calculated difference using the metric and the difference obtained through simulation is less than 6.58%. Our case studies also indicate that the developed Hungarian method based algorithm using the metric performs close to the optimal solution in comparison to random defect-redundant core assignments.**

I. Introduction

As technology advances, manycore architectures are becoming mainstreams for a large spectrum of applications, including real-time and embedded applications. As there are many cores on-chip, such architectures typically employ Network-on-Chip (NoC) as a scalable communication backbone among processing cores. However, many challenges are yet to be tackled for the design of NoC-based manycore processors. Manufacturing defects and transistor wear-outs are among the top list. 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 only 10% to 20%.

The research is supported by NSFC CNS 1018731, NSF Career 0746643, NSFC 60906018 and NSFC 61173006.

As there are many light weighted cores on-chip, and each core occupies only a small area of the chip footprint, core-level redundancy is often used as an efficient technique to overcome the NoC chip yield issue [2]. In particular, if C cores are expected to be provided to customers, R redundant cores will be added on the chip. Cores which fail due to defect or aging can be replaced with a redundant core, thus guaranteeing the demanded computing capability. However, when a defective core is replaced with a redundant core, it is possible that the on chip topology, i.e., the interconnect relationship among cores are changed, for example from a regular 2D mesh topology to a irregular topology. The underlying NoC topology with possible defective cores is called a physical topology. Different chips may have different physical topologies with different failure bitmaps. It will be a great burden for application developers as they have to face various topologies to design, deploy and optimize their programs. 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 purpose 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. For a real time application, rather than performance, the most important property is its predictability. That is, its functional and timing behavior should be as deterministic as system specifications require. Therefore, timing similarity instead of high performance is preferred to avoid introducing extra cost in redesign, re-implementing, retesting, and re-certifying the rest of the system when defect cores are replaced by redundant ones.

For a given application that has already been mapped to a manycore platform, if some of the cores on which application tasks are deployed become defective, with core-level redundancy, redundant cores are used to replace the defect cores. However, this reconfiguration will change the physical distance between two communicating tasks, which may further impact the application's timing behavior and cause timing variations.

Therefore, the question is how to select redundant candidates for the defect cores that minimize the timing variation.

When on a small scale NoC, i.e., both the number of redundant cores R and the number of defect cores D are small, we can traverse all the choices and find the optimal one offline with time cost of $O(R^D)$. However, in large scale manycore systems, for example, when there are thousands of cores per chip, the time cost is unaffordable even if it is computed offline.

In this paper, we focus on homogeneous manycore platforms and have made two major contributions: first, we have developed a metric that measures timing similarities between two different topologies upon which a real-time application is deployed. Second, we have developed a polynomial time algorithm to form a defect-free virtual topology that is most similar, with respect to the metric developed, to the application's initial reference topology.

The rest of the paper is structured as follows. Prior research is discussed in Section II. Section III discusses how to estimate timing behavior changes caused by defective core replacement. Section IV gives a brief background on generic Assignment Algorithm (also known as Hungarian method). Our proposed algorithm based on the Hungarian method to solve the replacement problem is presented in Section V. The evaluations of our work are shown through experiments in VI. Finally, we conclude and point out future work in Section VII.

II. Related Work

The NoC topology virtualization problem for *general purpose computing* has recently drawn great attention from the research community. Zhang et. al in [2], [3] studied the performance degradation of virtual topologies when compared to the topology initially designed. A heuristic approach called Row Rippling Column Stealing (RRCS) is proposed in [2] for homegenerous manycore processors. 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. They further extended the work to handle heterogeneous manycore processors [4].

Different optimization goals when reconfiguring the NoC topology have also been considered. For instance, Srinivasan [5] gives the $NMAP$ algorithm to minimize the average communication delay with the bandwidth constraints for NoC platforms; Hung [6] proposes IP virtualization and placement algorithm to achieve a thermally balanced design that minimizes temperature and energy consumption; and Hu [7] presents an efficient branch-and-bound algorithm to minimize the total communication energy through bandwidth reservation. As pointed out by Flich in [8] the three key metrics: performance, fault-tolerance (including yield), and power consumption are becoming the major concerns on the design of NoC systems. Research in this area have shown that topology virtualization is a promising technique to provide a unified solution to address all the three key issues.

Different from the prior work listed above, we focus on topology reconfiguration for applications with timing constraints. For these types of applications, rather than performance, the timing predictability and determinism are the paramount requirements. Hence, the reconfiguration objective is to use redundant cores provided on the chip to replace defective cores with the constraint of maximizing the timing resemblance of the newly configured topology to the initial one.

It is not difficult to see that the problem of finding appropriate redundant cores to replace defect cores belongs to the class of the assignment problems [9]. As a canonical solution to the assignment problem, the Hungarian method, which is first proposed by Harold [9] has been widely used. For instance, Jung-Hoon [10] uses it for solving resource allocation problem in mobile communication system and Gungor [11] applies it for multiple criteria assignment problems.

However, the topology reconfiguration problem with timing similarity constraint we are to address is not identical to the general assignment problem, in which, assignments are independent of each other and the cost of each assignment is fixed. In our problem, if more than one defective core exists, the cost of one's replacement may be dependent with others. We will address this issue in section V.

III. Timing Similarity between Two Virtual NoC Topologies for a given Real-time Application

In this section, we give a motivating example, define the timing similarity metric and give the problem formulation.

Before giving an example, we first define our application and NoC models.

Definition 1 (NoC): We represent the NoC as an array of physical cores under mesh topology and deterministic XY routing where

- $C = \{C_0, C_1, ..., C_M\}$ are the cores provided by the chip. We use $C_{(i)}$ and C_i to denote virtual core and physical core i respectively.
- $R = \{R_0, R_1, ..., R_N\}$ are the redundant cores provided by the chip.
- Virtual Topology $T : C \to C \cup R$ is an injective mapping from virtual cores to physical cores.
- $H^k_{(i),(j)}$ is the number of hops from $C_{(i)}$ to $C_{(j)}$ along the physical path of the traffic using XY routing algorithm for a given virtual topology T^k.

□

Definition 2 (Mapped Application Task Graph): A directed acyclic graph $\mathcal{A} = (J, E)$ mapped to a virtual topology where each vertex $j_i \in J$ represents the task mapped to virtual core $C_{(i)}$ and each edge $e_{ij} \in E$ is a data dependency between j_i and j_j. The volume of data sent along the directed edge e_{ij} is given by $v(e_{ij})$. □

We make two assumptions regarding the NoC. First, only the cores C become defective. Second, there is no contention in the network. In other words, the NoC bandwidth is sufficiently large. When there is no network contention, the hop count becomes the most important metric for communication times.

A. A Motivating Example

For a given 3x3 NoC with 3 redundant cores as shown in Fig.1(c). 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 mesh topology. There are several ways to do so, for instance, in Fig.1(c), the three redundant cores, R_0, R_1, or R_2, can be used to replace the defective core C_4, and hence generate three different virtual topologies. Even though applications are given a unified 3×3 mesh, different virtual topologies have different properties, such as latency and throughput.

For instance, given the application and its initial mapping shown in Fig.1(a), consider if the physical core C_4 is 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 between the virtual cores are different under XY routing(Table I). From the table it is clear that using R_0 as the virtual core C_4 is the best choice if the similarity metric is hop count.

Furthermore, if an application is deployed on a topology, the impact on the application of replacing a defective core by different redundant cores also varies.

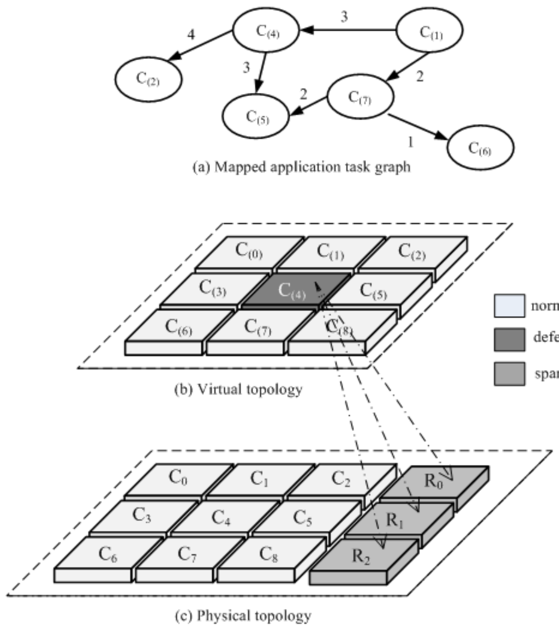

Fig. 1: Mapping application to cores

Generally, for a given application that is mapped to $|J|$ cores supported by a physical topology with $|R|$ redundant cores, if among the $|J|$ cores, p ($p \leq \min\{|J|, |R|\}$) cores become defective, to replace only those defective cores with redundant cores, we have $\binom{|R|}{p}p!$ number of different choices to form a new virtual topology. Then the question is: which one should we choose?

Traffic Between Virtual Cores	Corresponding Physical Cores	Hop Count
$C_{(1)} \to C_{(4)}$	$C_1 \to R_0$	2
$C_{(1)} \to C_{(4)}$	$C_1 \to R_1$	3
$C_{(1)} \to C_{(4)}$	$C_1 \to R_2$	4

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

B. Similarity Metric

In order to analyze the timing resemblance between a reference topology where there is no defect cores and a 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, processor speeds are the same. Hence, on-chip communication becomes a dominant factor that differentiates various virtual topologies' timing behavior.

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

Definition 3 (Traffic Time Delay): For a given application and virtual topology T^k, the traffic time delay from virtual core $C_{(i)}$ to core $C_{(j)}$ is defined as

$$F^k_{(i),(j)} = v(e_{ij}) + H^k_{(i),(j)} \tag{1}$$

□

For two different virtual topologies, T^k and T^r, their traffic time delay differences from virtual core $C_{(i)}$ to $C_{(j)}$ can be calculated by (2)

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

Clearly, the smaller the traffic time delay differences among *all* virtual core pairs, the higher the timing similarity among the two topologies. We use normalized average value (Ave) and normalized variation (Var) to model the traffic time delay difference of the entire topology. Their definitions are given below:

Definition 4: (*Normalized Average Traffic Time Delay Difference*)

Given an application, a new topology T^k and a reference topology T^r, the normalized average traffic time delay difference is given by (3)

$$Ave^k_r = \frac{\sum\limits_{e_{ij} \in E} \Delta^{k,r}_{(i),(j)}}{\Psi_r |E|} \tag{3}$$

where Ψ_r is the average traffic time delay of the reference topology T^r given by (4).

$$\Psi_r = \frac{\sum\limits_{e_{ij} \in E} F^r_{(i),(j)}}{|E|} \tag{4}$$

□

Definition 5: (*Normalized Average Variation of Traffic Time Delay Difference*)

Given an application, a new topology T^k and a reference topology T^r, the normalized variation of traffic time delay difference is given by (5)

$$Var_r^k = \sqrt{\frac{\sum_{e_{ij}\in E}(\frac{\Delta_{(i),(j)}^{k,r}}{\Psi_r} - Ave_r^k)^2}{|E|}} \tag{5}$$

where Ψ_r is given by (4).

□

Based on the traffic time delay difference between different topologies, we define virtual topology timing similarity as weighted sum of normalized average difference and variation. The formal definition is given below:

Definition 6: (*Virtual Topology Timing Similarity*)

Given an application, a new topology T^k and a reference topology T^r, their timing similarity $\chi(T_r^k)$ is defined by (6)

$$\chi(T_r^k) = w_a \times Ave_r^k + w_v \times Var_r^k \tag{6}$$

where w_a and w_v ($w_a + w_v = 1$) are the weights applied to the average difference and average variation, respectively.

□

C. Problem Formulation

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

Problem 1: For a given reference topology T^r on which a given application $\mathcal{A}$ is deployed, if there are cores used by the application $\mathcal{A}$ that become defective, construct a virtual topology T^{opt} with defective cores being replaced by redundant cores and satisfying the requirement (7)

$$\chi(T_r^{opt}) = \min\{\chi(T_r^k)\} \tag{7}$$

□

IV. The Hungarian Method

For self-containment, in this section, we briefly introduce the assignment problem and the Hungarian method that solves the assignment problem.

The assignment problem can be stated as [9]: there are n workers and k jobs, where $k \leq n$. Each job can only be assigned to a single worker. The time taken by each worker to finish a job is independent and can be different. The assignment problem is to find a worker to job assignment so that the total time taken to finish all the jobs is minimized.

The Hungarian method [9], [10], [11] solves the assignment problem in two steps:

Step 1 : constructs a cost matrix $M_{n\times n}$ where m_{ij} is the time cost for worker i ($1 \leq i \leq n$) to work on job j ($1 \leq j \leq k$). If $n \neq k$, i.e., the numbers of workers and the number of jobs are not the same, we augment the matrix M to a square one by adding $(n-k)$ number of columns filled with zeros.

Step 2 : uses equivalent matrix reduction to obtain the optimal assignment with respect to the cost matrix [9]

As an example, assume we have three workers W_1, W_2, and W_3, and three jobs J_1, J_2, and J_3. It takes W_1 25, 40, and 35 time units to finish job J_1, J_2, and J_3, respectively; for W_2, 40, 60, and 35 and W_3, 20, 40, and 25 to finish the three jobs, respectively. We have the cost matrix $M_{3\times3}$ as:

$$M_{3\times3} = \begin{bmatrix} 25 & 40 & 35 \\ 40 & 60 & 35 \\ 20 & 40 & 25 \end{bmatrix}$$

By iteratively reducing the value in rows and columns, the Hungarian method is able to get the optimal assignment in $O(n^3)$ time [12].

For the given cost matrix, the reduced matrix of running the Hungarian Algorithm is

$$M_{3\times3} = \begin{bmatrix} 0 & \mathbf{0} & 10 \\ 5 & 10 & \mathbf{0} \\ \mathbf{0} & 5 & 5 \end{bmatrix}$$

Hence, the optimal assignment is W_1, W_2, and W_3 for J_2, J_3, and J_1, respectively.

V. Hungarian Method Based Virtulization Algorithm

As formulated in Section III-C, we need to assign appropriate redundant cores to replace defect cores for a given application so that the communication time change among tasks is minimized. It is not difficult to see that, it is also a type of assignment problem.

Recall that the first step in using the Hungarian method to solve an assignment problem is to form a cost matrix. It is worth pointing out that in the worker/job assignment problem, the time cost for a worker i to perform a job j is fixed and independent from how the job is assigned to other workers.

Hence, in order to use the Hungarian method, we have to first construct a cost matrix. For defect-redundant core assignment problem, the related cost of replacing defective core d with redundant core r , i.e., M_{dr} is the communication time delay difference among all the virtual communicating cores. Unfortunately, if there is more than one defective core to be replaced, we will not be able to construct the matrix due to lack of information in computing the traffic delay difference χ given by (6).

For example, if physical cores C_4, C_5 and C_7 in Fig. 1 are defective then redundant cores R_0, R_1, R_2 can be used replace them. We need to build up a 3×3 matrix, and the entry in the first row and first column indicating the timing difference when using physical core R_0 to replace physical defective core C_4, which needs the traffic time delay between virtual core $C_{(7)}$ and $C_{(5)}$ after reconfiguration. Unfortunately, these values are not known until we know which redundant cores are assigned to the defective cores.

One way to overcome the problem is to assume that there is only one defective core at a time. In other words, when deciding the cost values for replacing defective core f, we treat the other defective cores (if any) as normal defect-free cores. Using the above example, when filling the entry indicating physical core R_0 to replace defective core C_4, we assume physical cores C_7 and C_5 are not defective. Once the cost matrix with respect to communication time delay differences is obtained, we can use the Hungarian method to find the optimal replacement for the defective cores. The Hungarian method based virtualization (HMBV) algorithm is given in Algorithm 1.

Algorithm 1 `HMBV Algorithm(`T^r`)`

```
1: for each defective core $C_i$ do
2:     for each redundant core $R_j$ do
3:         calculate $\chi(T_r^i)$
           where $T^i$ maps core $C_{(i)}$ to $R_j$
4:         $M_{ij} \leftarrow \chi(T_i^j)$
5:     end for
6: end for
7: apply Hungarian Algorithm to obtain the solution for cost
   matrix $M$
```

The steps $1-6$ are to construct the cost matrix which has time complexity of $O(n^2)$, and step 7 uses the general Hungarian algorithm to find the defect-redundant mapping solution which is of time complexity $O(n^3)$. Therefore, the time complexity for the algorithm is $O(n^3)$.

Let us use the example shown in Fig. 1 to illustrate the algorithm. Again, assume the core C_4, C_5 and C_7 are defective, steps $1-6$ in algorithm 1 generate the cost matrix

$$M_{3\times3} = \left[\begin{array}{c|ccc} Defect\ \ Core & R_0 & R_1 & R_2 \\ \hline C_4 & 0.2 & 0.2727 & 0.3333 \\ C_5 & 0.0929 & 0.1245 & 0.1899 \\ C_7 & 0.1111 & 0.1724 & 0.2429 \end{array}\right]$$

Apply the Hungarian method to the cost matrix, we have that the new configuration, where R_0 is used to replace C_4, R_2 for C_5, and R_1 for C_7, respectively, most resembles to the initial configuration with respect to the communication time delays among communicating tasks.

Discussion

Admittedly, the cost matrix generated under the assumption, i.e., when computing the cost values for a specific defective core, all other defective cores are treated as defect-free, may not reflect the whole property of the problem we are to solve. It is not difficult to see that the defect-redundant core assignment problem in its original form is of time complexity $O(|R|^D)$, where D and $|R|$ are the number of defect cores and redundant cores, respectively. The HMBV algorithm of complexity $O(|R|^3)$, hence, does not generate the optimal solution for our defect-redundant core assignment problem. In the next section, we will empirically evaluate both the similarity metric and the HMBV algorithm.

VI. Evaluation

In this section, we perform two sets of experiments. First we evaluate the metric given in (1) by comparing it with the simulated communication time. Second we evaluate the remapping obtained by the Hungarian algorithm by comparing it to both the optimal solution and a randomly generated solution.

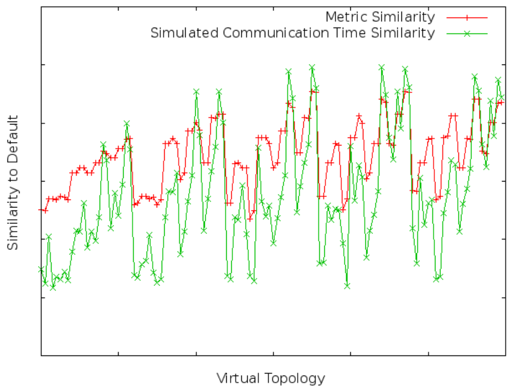

Fig. 2: Model Accuracy

A. Experiment Setup

NIRGAM (NoC Interconnect Routing and Application Modeling) [13] is a modular and cycle accurate simulator developed in SystemC. In NIRGAM, a 2D NoC 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. Wormhole switching and deterministic XY routing are used on the mesh.

In our experiments we use TGFF[14] to generate a task graph and create a random mapping of the task to a 5x5 mesh. The mapping is random because we do not assume anything about the original mapping of the application and only aim to achieve similarity. We assume that there is an extra column of cores located on the right side of the chip to provide n redundant cores for an $n \times n$ virtual topology. The location and number of the redundant cores is not important however as the reconfiguration algorithm is aware of their locations.

To satisfy the assumption that no contention exists in the network, the flit injection rate of each task is sufficiently low so that each router can handle all of the traffic in the application without contention.

B. Metric Evaluation

In this experiment we show that the metric given in (1) is a good model for the communication time in the application. For an edge in a application $e_{ij} \in E$ with data volume $v(e_{ij})$ the communication time is the amount of time from when j_i begins transmission of data to when j_j receives the last flit in $v(e_{ij})$. Since the NoC is homogeneous the execution time is the same on any core. Therefore with similar communication times along every edge the start and finish time of tasks will be similar.

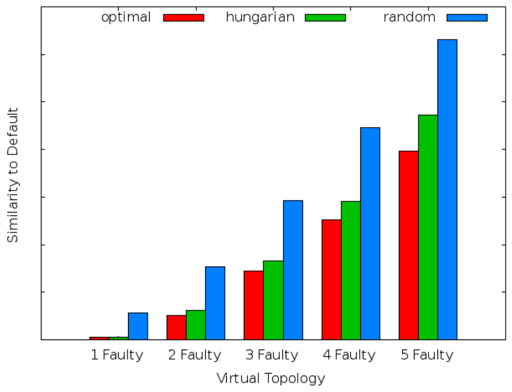

Fig. 3: Algorithm Evaluation

Fig. 2 shows the entire solution space i.e. all T^r for a specific set of defective cores for a specific application. The line *Metric Similarity* is calculated using (6) whereas the line *Simulated Communication Time Similarity* uses (6) but subsitutes the actual simulated communication time using NIRGAM for (1). The standard deviation of the difference between these two data sets is 6.58%.

C. Algorithm Evaluation

In this experiment we compare the HMBV algorithm to both the optimal solution and a random solution. For a specific application we chose five random defective core sets for each number of possible defective cores. For each defective set we find the optimal solution, the HMBV solution and a random solution. For each solution we calculate its similarity to the defect-free mapping. As can be seen from Fig. 3, the HMBV algorithm. performs close to the optimal solution in comparison to a random i.e. average solution.

VII. Conclusion

Virtualization in manycore systems in presence of manufacturing defects and device wear-outs for real time and embedded applications is very complex and it depends heavily on the desired hardware architecture, timing requirements, and the on-chip redundancy distributions. By taking these factors into considerations, the developed method based on Hungarian algorithm allows us to find the virtual topology that is most similar, in terms of their timing behaviors captured by their task-to-task communication times, to the reference topology an application is initially designed on. The simulation results show the effectiveness of our approach.

The research presented in the paper is only the first step toward applying virtualization technologies to real-time and embedded applications. We are all aware that for hard real-time applications, minimizing task-to-task communication time change may not guarantee stringent timing properties required by these applications. Our immediate next steps are to remove our assumption of no contention in the network, to study how reconfiguration may impact hard real-time applications, and investigate virtualization techniques that guarantee deadline satisfactions. Real-time and embedded applications often have other resource constraints, such as peak temperature and energy consumption constraints, the virtualization problem becomes more challenging when these concerns must be taken into consideration. This is another area of our future study. Further extension to the work is to address these concerns for heterogeneous manycore systems.

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] G. D. M. Srinivasan Murali, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," in *Proceedings of the conference on Design, automation and test in Europe*, 2004, pp. 1530–1591.

[6] W. Hung, C. Addo-Quaye, T. Theocharides, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Thermal-aware ip virtualization and placement for networks-on-chip architecture," *Computer Design, International Conference on*, vol. 0, pp. 430–437, 2004.

[7] R. M. J. Hu, "Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints," in *Proceedings of the ASP-DAC 2003*, ser. CODES/CASHE '98, 2003, pp. 233–239.

[8] J. Flich, S. Rodrigo, J. Duato, T. Sodring, A. Solheim, T. Skeie, and O. Lysne, "On the potential of noc virtualization for multicore chips," *Complex, Intelligent and Software Intensive Systems, International Conference*, vol. 0, pp. 801–807, 2008.

[9] H. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[10] O. Jung-Hoon, Noh. Seong-Jun, "Distributed SC-FDMA Resource Allocation Algorithm Based on the Hungarian Method," in *2009 70th IEEE on Vehicular Technology Conference*, 2009, pp. 1–5.

[11] M. Gungor, I. Gunes, "Fuzzy multiple criteria assignment problems for fusion: the case of Hungarian algorithm," in *2000 3th IEEE International Conference on Information Fusion*, 2000, pp. 408–412.

[12] J. Munkres, "Algorithms for the Assignment and Transportation Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[13] 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.

[14] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Proceedings of the 6th international workshop on Hardware/software codesign*, ser. CODES/CASHE '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 97–101.