Maintaining Real-Time Application Timing Similarity For Defect-Tolerant NoC-based Many-Core Systems

Zheng Li, Illinois Institute of Technology
Frank Lockom, Illinois Institute of Technology
Shangping Ren, Illinois Institute of Technology

Many-core Network-on-Chip (NoC) processors are emerging in broad application areas, including those with timing requirements, such as real-time and multimedia applications. Typically, these processors employ core-level backup to improve yield. However, when defective cores are replaced by backup ones, the NoC topology changes. Consequently, a fine-tuned application based on timing parameters given by one topology may not meet the expected timing behavior under the new one. We first develop a metric to measure timing similarity of an application on different NoC topologies and then propose mixed binary quadratic programming and greedy algorithms to reconfigure a defect-tolerant many-core NoC.

Categories and Subject Descriptors: C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms: Network on Chip, Many-core, Virtualization

Additional Key Words and Phrases: Timing similarity, Multimedia application, Guaranteed service, Mixed binary quadratic programming

ACM Reference Format:
Zheng Li, Frank Lockom, and Shangping Ren, 2013. Maintaining Real-Time Application Timing Similarity For Defect-Tolerant NoC-based Many-Core Systems. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (January YYYY), 20 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

As technology advances, many-core architectures are becoming the mainstream for a large spectrum of applications, including real-time applications and multimedia applications. As there are many cores on-chip, such architectures typically employ Network-on-Chip (NoC) as a scalable communication backbone among processing cores [Dally and Towles 2001; Benini and De Micheli 2002]. However, many challenges are yet to be tackled for the design of NoC-based many-core processors, manufacturing defects and transistor wear-outs are among the top list. According to Sperling's report [Sperling 2007], 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%.

As there are many light weighted cores on-chip, and each core occupies only a small area of the chip footprint, core-level backup is often used as an efficient technique to overcome the NoC yield issue [Zhang et al. 2008]. Cores which become defective due to the manufacturing process or wear-out can be replaced with a backup core, thus guaranteeing the demanded computing capability. However, when a defective core is

This work is supported by the NSF CNS 1018731 and NSF Career 0746643.

replaced with a backup core, it is possible that, the physical topology, i.e., the interconnect relationship among cores is changed.

Different physical topologies may have different failure maps with different performance characteristics, it would be a great burden for application developers to consider the various topologies to deploy and optimize their applications. Topology virtualization is proposed to isolate the variation of the underlying physical structure, and provide application developers with the same virtual topology of a defect-free chip [Zhang et al. 2009].

Prior research on many-core topology virtualization mainly focused on the general purpose computing domain and the methods proposed intend to achieve better performance in terms of communication latency and network throughput [Zhang et al. 2008]. However, for a real-time application, rather than performance, the most important property is the timing predictability. In order to avoid introducing extra cost in redesigning, re-implementing, retesting, and re-certifying the system during the reconfiguration, the timing behaviors of the application after reconfiguration should be similar to the one on a defect-free chip.

For a given application that has already been mapped to a NoC platform, if some of the cores on which tasks are deployed become defective, with core-level backup, backup cores are used to replace the defective cores. However, this reconfiguration will change the physical distance between two communicating tasks, which may impact the application's timing behaviors and cause timing variations. Therefore, the question is how to select backup candidates to minimize the timing variation.

When on a small scale NoC, i.e., both the number of backup cores B and the number of defective cores F are small, we can traverse all $B!/(B-F)!$ choices for remapping the defective cores and find the optimal one. However, in a large scale NoC, when there are hundreds of cores per chip with B and F large enough, the time cost is unaffordable even if it is computed offline.

In this paper, we will focus on the homogeneous 2-dimensional (2D) mesh many-core NoC platforms to develop a metric to measure the timing similarity between two different topologies upon which a real-time application is deployed. With respect to this metric, we will provide two solutions: a mixed binary quadratic programming (MBQP) approach and a greedy algorithm. The MBQP approach can get the optimal solution by using commercially available tools to solve the reconfiguration problem. However, the MBQP approach is time consuming and may only be acceptable for offline reconfiguration. The greedy algorithm is recommended for online reconfiguration. Although the optimal solution is not guaranteed, experiment results have shown that the greedy choice averages among the top 10% of all the possible choices.

The rest of the paper is organized as follows. Related work is discussed in Section 2. We give our NoC and application model in Section 3. In Section 4, we first provide a motivating example, continue by developing our timing similarity metric and then use this metric to formulate the problem we are to solve. Our proposed algorithms and a discussion of our previous work are then given in Section 5 and are evaluated in Section 6. Finally we conclude in Section 7.

2. RELATED WORK

As mobile computing becomes pervasive, NoC platforms have been developed to accommodate real-time and multimedia applications. Much of the research has been done in the area of NoC topology explorations. For instance, Lankes et al. [2009] studied a NoC topology exploration based on a real-world mobile multimedia application 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. [2010] conducted system-level exploration of mesh-based NoC architectures for multimedia applications. They established a simulation platform and proposed an exploration approach to obtain the optimal design for specific applications.

Optimization of multimedia application to NoC platform mapping problems have also been studied extensively. Haiyun et al. [2007] addressed a mapping algorithm of irregular mesh NoC for portable multimedia applications to achieve the minimum communication cost with certain constrains. Chou and Marculescu [2008] analyzed the impact of network contention on the application mapping for NoC architecture and then formulate the contention-aware application mapping problem which aims at minimizing the network contention to an integer linear programming problem. Derin et al. [2011] focused on minimizing the communication traffic in the system and the total execution time of the application to do the application to NoC platform mapping. They also formulated this problem to an integer linear programming problem.

Virtualization provides a unified hardware interface for applications. The topology virtualization problem for general purpose computing is discussed thoroughly in [Zhang et al. 2008, 2009, 2010]. 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 approach called Row Rippling Column Stealing (RRCS) is proposed in [Zhang et al. 2008]. The essence of RRCS is to maintain the physical regularity of reconfigured virtual topologies in both row and column units, and hence to maximize performance. Unlike our approach, RRCS is application agnostic. We evaluate its performance in section 6.

Rather than using performance as a single objective, many-core topology virtualization for applications with specific timing requirements, such as multimedia application containing real-time data, also needs to consider maintaining timing similarities among different topologies. Many notions of timing behavior similarities have been proposed in various literature. For instance, Henzinger et al. [2005] defined quantitative notions of timed similarity and bisimilarity on timed systems. They also gave algorithms to compute the distance between two timed systems modeled as timed automaton. Recent research shows that the timing behavior of a system can be characterized by a feasible region defined by the system's timing constraint set [Yu et al. 2010].

In addition, the prevalence of multimedia applications also advanced the evolution of the physical architecture of NoC. In a conventional NoC platform, packages need to compete for routing and transmitting on link, therefore, if congestion appears, the transmission time delay is unpredictable. But multimedia applications always contain real-time data, which require guaranteed performance to meet the required levels of service. Goossens et al. [2005] developed the Æthereal NoC by pipelined time-division-multiplexed circuit switching, which can provide guaranteed service by time slot reservation. However, this architecture has the scalability issue, because of the global synchronicity requirement, that is, all the routers in the network must occupy the same fixed-duration slot. Based on globally asynchronous locally synchronous concept, Bjerregaard and Sparso [2005] developed the MANGO clockless NoC, which can provide guaranteed service by virtual channel reservation.

As guaranteed service based NoC is the trend for multimedia and real-time applications, in this paper, we will consider this type of architecture to analyze the timing similarity problem during the topology reconfiguration. The objective is to use backup cores provided on the chip to replace defective cores and at the same time to maximize the timing behaviors resemblance of the application on the newly configured one to that on the initially designed one.

3. SYSTEM MODEL

Before formulating our problem we define our system model, we also define the notations which are to be used throughout this article.

3.1. Network On Chip (NoC) Many-Core Processor Model

Our many-core NoC model is based on a 2D mesh topology with homogeneous cores under XY routing. In a 2D mesh topology, cores are arranged in rows and columns and are connected to (at most four) adjacent cores via bidirectional links. The chip provides a *virtual topology* of cores to applications which is supported by an underlying physical topology with an additional set of backup cores.

As our focus is on real-time applications, we assume the NoC provides guaranteed service (latency and throughput) for the applications. Guaranteed service is provided by reserving a portion of the bandwidth available on a link so long as the total reserved bandwidth does not exceed 1. This can be seen as a generalization of time division multiplexed circuit switching in Æthereal [Goossens et al. 2005] or virtual channel reservation in MANGO [Bjerregaard and Sparso 2005]

The NoC is represented by the undirected graph (C, L) where each $c_i \in C$ represents a physical core in the NoC and each edge $l_k = (c_i, c_j) \in L$ is a direct communication link between two adjacent physical cores. The NoC takes the form of a 2D mesh topology as in Fig. 1(a). Additionally we have:

— $\widetilde{C}$ denotes the set of virtual cores which are provided to applications by the NoC, where $|\widetilde{C}| < |C|$ and $c_{(i)} \in \widetilde{C}$ is an individual virtual core.
— B is the set of backup cores provided by the chip where $B \subset C$, $|B| = |C| - |\widetilde{C}|$ and $b_i \in B$ denotes an individual backup core.
— A *virtual topology mapping* [1] $M^k_{\widetilde{C}C} : \widetilde{C} \to C$ is an injective function mapping from each virtual core to a unique physical core. We represent $M^k_{\widetilde{C}C}$ as a logical matrix where $M^k_{\widetilde{C}C}(i,j) = 1$ indicates $c_{(i)} \in \widetilde{C}$ is mapped to $c_j \in C$. The *defect-free* or *reference* mapping is the mapping when no cores are defective and is given by $M^{ref}_{\widetilde{C}C} : \widetilde{C} \to C \backslash B$. Changing the virtual topology mapping is called *reconfiguration*. We use $M^{new}_{\widetilde{C}C}$ to denote the mapping after reconfiguration and $M^k_{\widetilde{C}C}$ to denote a generic mapping.
— Defective physical cores are denoted by the set $F \subset C$. The set $\widetilde{F} \subseteq \widetilde{C}$, denotes the corresponding affected virtual cores given by the reference mapping, these are the cores which must be remapped.
— Given the set of affected virtual cores $\widetilde{F}$, our problem is to construct a new virtual topology mapping $M^{new}_{\widetilde{C}C}$ which remaps the affected virtual cores to backup cores. More precisely, we would like to find the value of each binary variable $x_{i,j}$ in

[1]Throughout this paper, we use the notation M^i_{AB} which denotes a relation $M^i : A \to B$ represented as a logic matrix between two previously defined sets A, B. The variable i is only used to differentiate relations of the same form. Occasionally it is also used to represent a relation of the form $M^i : (A, B) \to \mathbb{Z}$ or some other number set.

$$M^{new}_{\widetilde{C}C}(i,j) = \begin{cases} M^{ref}_{\widetilde{C}C}(i,j), & c_{(i)} \notin \widetilde{F} \\ x_{i,j}, & c_{(i)} \in \widetilde{F}, c_j \in B \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

This indicates that we only change the mapping of defective cores and do not consider a full remapping. It is possible that a full remapping may provide a better solution however we avoid a full remapping for two reasons. First, during online reconfiguration, there will be a migration cost associated with each remapping. Only considering movement of defective cores will keep this to a minimum. Second, we would like to avoid other variations not accounted for in our model such as different thermal or power consumption properties which may vary over the NoC. Thus we aim to find a solution which utilizes the original mapping as a starting point and makes only necessary changes.

3.2. Mapped application task graph

Applications in this article are represented by task graphs similar to those used in [Dick et al. 1998; Liu 2000; Lei and Kumar 2003; Chou and Marculescu 2008] which are mapped to a virtual topology. The DAG (V, E) represents an application where each $v_i \in V$ is a task and each edge $e_i = (v_i, v_j) \in E$ represents a data dependency of v_j on v_i. In addition, we have

— Each task v_i is mapped to a virtual core in $\widetilde{C}$ given by $M_{V\widetilde{C}} : V \rightarrow \widetilde{C}$. Note that this mapping is fixed and does not change during reconfiguration. $M_{V\widetilde{C}}$ is represented as a logical matrix.
— Each task has an *execution time* in cycles given by $exec(v_i)$, $exec$ will be used as a column vector in matrix operations.
— Each edge has a *data volume* in flits given by $vol(e_i)$, vol will also be used as a column vector.
— Each edge has an *injection rate* in cycles/flit given by $rate(e_i)$. The reciprocal of injection rate is bandwidth given by $bandwidth(e_i)$. Both $rate$ and $bandwidth$ are also used as column vectors.

Given an NoC and application task graph the most basic property which determines timing similarity is the hop count between communicating tasks. The hop count of a communication edge $e_k = (v_i, v_j)$ is the number of links over which data will travel from v_i to v_j under XY routing. We use the logical matrix $M^k_{EL} : E \rightarrow L$ to represent the links which are used by an edge for a given virtual topology mapping, $M^k_{\widetilde{C}C}$. Thus we can define the hop count $hops^k : E \rightarrow \mathbb{N}$ for each edge as

$$hops^k = M^k_{EL} \times 1_{|L|} \tag{2}$$

where $\mathbb{N}$ is the set of natural numbers and $1_{|L|}$ is the column vector of 1's of size $|L|$. Calculation of M^k_{EL} is left to section 5.1.

We assume that $M_{V\widetilde{C}}$ does not overload any link in the defect free topology under XY routing. A link is overloaded if the total bandwidth allocated across it is greater than 1. It is important to note that when reconfiguring the virtual topology, we must ensure that no link becomes overloaded. This constraint can be represented as $(M^k_{EL})^T \times bandwidth \leq 1_{|L|}$, where $(M^k_{EL})^T$ is the transpose of M^k_{EL} and $bandwidth$ is the column vector of the fraction of a link's bandwidth required by each communication edge. This implies we are not concerned with *when* a link may be in use. If a communication edge utilizes a link it must reserve its required portion of bandwidth at all

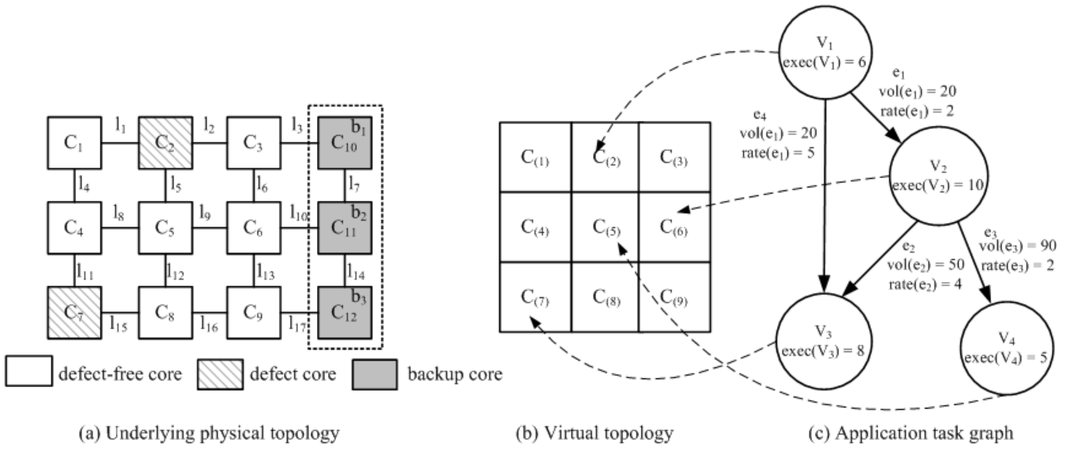

Fig. 1. Example application and NoC

times. This assumption is consistent with the mechanisms by which guaranteed service is provided in the NoC architectures previously mentioned [Goossens et al. 2005; Bjerregaard and Sparso 2005].

The execution of an application task graph is as follows: A task v_i begins execution when it has received the required data volume from all its parents, i.e., $\{vol(v_k)|(v_k, v_i) \in E\}$. After $exec(v_i)$ cycles the task is completed and begins sending the required data volume at the injection rate required to all of its children. After a task has finished communication, it will again begin execution after satisfying its dependencies; or if it has no dependencies, execution will begin immediately.

4. PROBLEM FORMULATION

Before developing our timing similarity metric and formulating our problem, we first give a brief example to help solidify our models and motivate the problem.

4.1. Motivating Example

Suppose we have the application shown in Fig. 1(c). It is mapped to the 3×3 virtual topology given in Fig. 1(b) and is supported by the underlying physical topology providing 3 backup cores located on the right most column as pictured in Fig. 1(a). Now suppose that the physical core c_6 becomes defective, we must create a new virtual topology mapping, $M^{new}_{\widetilde{C}C}$, by changing the mapping of $c_{(6)}$ on the defect-free topology to either b_1, b_2 or b_3.

Reconfiguration may change the timing behavior of the application by changing the hop count of a communication edge. As task v_2 is mapped to $c_{(6)}$, the hop count of all edges in the application incident with v_2 must be examined. The resulting hop counts are listed in Table I.

Additionally we must ensure that the remapping does not cause any of the links to be overloaded. For example if $M^{new}_{\widetilde{C}C}(c_{(6)}) = b_1$, links l_2 and l_3 will be overloaded, because they would be on the paths corresponding to e_1, e_2 and e_3 and the total bandwidth exceeds the links' capacity($1/2 + 1/4 + 1/2 > 1$). Therefore b_1 can not be the candidate.

In order to minimize the impact to the existing system and save the extra cost for redesign, re-implementing, retesting and re-certifying the system, we should keep the application's timing behaviors after reconfiguration as similar to the original one as

Table I Hop counts for different reconfigurations of $c_{(6)}$

	e_1	e_2	e_3
$M^{ref}_{\widetilde{C}C}$	2	3	1
$M^{new}_{\widetilde{C}C}(c_{(6)}) = b_1$	2	5	3
$M^{new}_{\widetilde{C}C}(c_{(6)}) = b_2$	3	4	2
$M^{new}_{\widetilde{C}C}(c_{(6)}) = b_3$	4	3	3

possible. If our metric for timing similarity was average hop count of each edge then our choice would be $M^{new}_{\widetilde{C}C}(c_{(6)}) = b_2$.

Simply looking at the hopcount may not always yield the best result however. Considering our example, if the hop count of e_1 changes, the *communication time* will change and the *starting time* of v_2 will also be changed. This change will then propagate to the starting time of v_3 and v_4, in addition to the changes introduced by e_2 and e_3, respectively. Based on the analysis above, we will formulate our timing similarity metric in the next section.

4.2. Timing Similarity Metric

As we assume the many-core systems are homogeneous, the tasks execution times will not vary due to reconfiguration. Hence, on-chip communication is the dominant factor that differentiates various timing behaviors. We define communication time below.

Definition 4.1. Communication Time:
Given a virtual topology mapping, $M^k_{\widetilde{C}C}$, and an application task graph, (V, E), the communication time in cycles for each edge is given by the column vector $comm^k : E \to \mathbb{N}$:

$$comm^k = hops^k + diag(rate) \times diag(vol) \times 1_{|E|} \tag{3}$$

where $\mathbb{N}$ is the set of natural numbers, $hops^k$ is given by Eq. (2) and $diag(vector)$ is the diagonal matrix formed using $vector$.

□

According to the definition of an application task graph, a task can only start when the required data from all its precedences has arrived. Thus we can define the starting time of a task recursively given the starting time, execution time, and communication time of all its parents. Note that the following definitions are not circular as we assume the mapped application task graph is a DAG.

Definition 4.2. Starting Time:
Given a virtual topology mapping $M^k_{\widetilde{C}C}$, the starting time in cycles of a task is defined by $start^k$

$$start^k(v_i) = max\{M^k_{VV}(i, :)\} \tag{4}$$

where M^k_{VV} is a $|V| \times |V|$ matrix such that $M^k_{VV}(i, j) = n$ indicates that v_j is a precedent of v_i in the application task graph and that v_i has received all of the data from v_j at time n. $max\{M^k_{VV}(i, :)\}$ indicates the largest element in row i.

$$M^k_{VV}(i, j) = \begin{cases} start^k(v_j) + exec(v_j) + comm^k((v_j, v_i)) \text{ if } (v_j, v_i) \in E \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

□

For our model we conclude that if after reconfiguration, the starting time of all tasks remains the same then the timing properties are the same.

In statistics, *Euclidean Distance* is often used to measure the distance between the sample spaces of random variables, which is defined as:

$$D_{Q_1,Q_2}(2) = \sqrt{\sum_{i=1}^{n} |f_{Q_{1,i}} - f_{Q_{2,i}}|^2} \tag{6}$$

where $f_{Q_{j,i}}$ is the sampled value of variable i in sample space Q_j and n is the number of variables.

In most cases, distance and similarity measures are interchangeable in the sense that a small distance means high similarity. Based on this sense, the $Euclidean\ Distance$ has been widely used to measure similarity between two images [Hastie et al. 2008]. Similarly, we map our time similarity problem to the distance problem in the following way: the starting time of each task is treated as a random variable, and all tasks' starting times on the defect-free virtual topology mapping $M^{ref}_{\widetilde{C}C}$ and the one after reconfiguration $M^{new}_{\widetilde{C}C}$ can be treated as different sample spaces, so the timing similarity can be defined as:

$$D_{new} = \sqrt{\sum_{v_i \in V} (start^{ref}(v_i) - start^{new}(v_i))^2} \tag{7}$$

4.3. Problem Formulation

With the metric of timing similarity defined as Eq. (7), we can now formulate our problem as:

Objective: Minimizing the timing distance.
Given an application task graph (V, E), the NoC (C, L), which the application is mapped to and a set of defective cores $F \subset C \setminus B, |F| \leq |B|$, assign a value to each binary variable $x_{i,j}$ in Eq. (1) such that:

$$minimize \;\; D_{new} \tag{8}$$

Subject to:
Constraint 1: The bandwidth across each link must not exceed its capacity.

$$(M^{k}_{EL})^T \times bandwidth \leq 1_{|L|} \tag{9}$$

Constraint 2: Each virtual core is mapped to only one physical core.

$$M^{new}_{\widetilde{C}C} \times 1_{|C|} = 1_{|\widetilde{C}|} \tag{10}$$

Constraint 3: At most one virtual core can be mapped to a given physical core.

$$(M^{new}_{\widetilde{C}C})^T \times 1_{|\widetilde{C}|} \leq 1_{|C|} \tag{11}$$

As we have mentioned, when the number of backup cores B and the defective cores F are large enough, enumerating all $B!/(B-F)!$ permutations is not feasible. In the next section we will give our two proposed algorithms. First we will transform this optimization problem to a mixed binary quadratic programming problem, which can be solved by commercial tools, such as IBM ILOG CPLEX [IBM 2008]. Second, we give a greedy solution which may not find the optimal solution but can finish in polynomial time under certain constrains to be specified.

5. MAINTAINING APPLICATION TIMING SIMILARITY

In this section, we present two approaches to address the problem of maximizing timing similarity defined in section 4.3. We also discuss our previous work on this topic.

5.1. Mixed Binary Quadratic Programming (MBQP) Approach

5.1.1 General MBQP

Mixed binary quadratic programming problem is a special type of mixed integer quadratic programming problem. It contains a quadratic function subject to linear constraints on its variables, and some of the variables are constrained to binary values. The mathematical definition can be found in [Axehill 2005]. Although the MBQP is a well known NP-hard problem [Wolsey 1998], some commercial tools, such as IBM ILOG CPLEX, are available to solve it efficiently [IBM 2008]. Unfortunately, in our formulated problem, the objective function in formula (8) is not quadratic and the constraint in formula (9) is not represented in linear format. Next, we discuss how to transform the maximizing timing similarity problem to the MBQP problem.

5.1.2 linear calculation of M_{EL}^{new} and M_{VV}^{new}

In order to obtain a quadratic objective function[2] and represent the constraint (9) in linear format, which is required by MBQP, the following issues must be resolved in order to complete the formulation:

— represent M_{EL}^{new} and M_{VV}^{new} in linear format to get starting time in formula (4) and the bandwidth constraint in formula (9).
— transform the max function in formula (4) to a set of linear constraints.

We first introduce the concept of paths in the NoC. For any two physical cores (c_i, c_j) in the NoC, there is only one set of links on which data will travel between them under deterministic XY routing. We call this set of links a path. As these paths are based on the physical topology of the NoC, they are statically defined and do not change for different virtual topology mappings. We call the set of all paths P, where $|P| = |C|^2$. We define three matrices associated with paths.

— $M_{PL} : P \to L$ where $M_{PL}(i,j) = 1$ indicates $l_j \in L$ is part of path p_i.
— $M_{CP_s} : C \to P$ where $M_{CP_s}(i,j) = 1$ indicates $c_i \in C$ is the *source* of path p_j.
— $M_{CP_k} : C \to P$ where $M_{CP_k}(i,j) = 1$ indicates $c_i \in C$ is the *sink* of path p_j.

In order to define the M_{EL}^{new} and M_{VV}^{new}, the information of application task graph is also needed. We use the following two matrices to represent the application task graph, which are independent of different virtual topology mappings.

— $M_{EV_s} : E \to V$ where $M_{EV_s}(i,j) = 1$ indicates $v_j \in V$ is the *source* of edge $e_i \in E$.
— $M_{EV_k} : E \to V$ where $M_{EV_k}(i,j) = 1$ indicates $v_j \in V$ is the *sink* of edge $e_i \in E$.

As defined in section 3.2, $M_{EL}^{new} : E \to L$ is to represent the mapping from edges to links. In order to get this mapping, we must first map edges to paths. Hence, we build two auxiliary matrices:

— $M_{EP_s}^{new} : E \to P$ where $M_{EP_s}^{new}(i,j) = 1$ indicates the physical core that the *source* of edge e_i mapped to is the *source* core of path p_j under the new virtual topology.

[2]When specifying the MBQP problem, we do not apply the square root operation in the object function (7), as it must be quadratic.

— $M_{EP_k}^{new}: E \rightarrow P$ where $M_{EP_k}^{new}(i,j) = 1$ indicates the physical core that the *sink* of edge e_i mapped to is the *sink* core of path p_j under the new virtual topology.

Using the introduced matrices above, we can represent $M_{EP_s}^{new}$ and $M_{EP_k}^{new}$ as:

$$M_{EP_s}^{new} = M_{EV_s} \times M_{V\widetilde{C}} \times M_{\widetilde{C}C}^{new} \times M_{CP_s}$$

and

$$M_{EP_k}^{new} = M_{EV_k} \times M_{V\widetilde{C}} \times M_{\widetilde{C}C}^{new} \times M_{CP_k}$$

An edge is mapped to a path if and only if *both* the source and sink physical cores of that edge correspond to the path. Thus the edges to paths mapping can be represented as:

$$M_{EP}^{new}(i,j) = min(M_{EP_s}^{new}(i,j), M_{EP_k}^{new}(i,j)) \quad (12)$$

which can be represented by the following set of linear constrains:

$$\begin{aligned} &M_{EP}^{new}(i,j) \leq M_{EP_s}^{new}(i,j), \\ &M_{EP}^{new}(i,j) \leq M_{EP_k}^{new}(i,j), \\ &M_{EP}^{new}(i,j) \geq M_{EP_s}^{new}(i,j) + M_{EP_k}^{new}(i,j) - 1 \end{aligned}$$

Finally, we are in a position to define $M_{EL}^{new}: E \rightarrow L$

$$M_{EL}^{new} = M_{EP}^{new} \times M_{PL} \quad (13)$$

Now we are ready to re-define M_{VV}^{new} in the linear format. Recall from its previous definition in Eq. (5) that the i^{th} row in M_{VV}^{new} contains the time at which v_i's parents have finished sending data to v_i, i.e., starting time + execution time + communication time. With this in mind we have:

$$M_{VV}^{new} = (M_{EV_k})^T \times [M_{EV_s} \times (diag(start^{new}) + diag(exec)) + diag(comm^{new}) \times (M_{EV_s})] \quad (14)$$

Lastly, we need to formulate the max function in formula (4), which is in the format of $x_{max} = max\{x_1, x_2, ..., x_n\}$, to a set of linear constraints. When $n = 2$, that is, $x_{max} = max\{x_1, x_2\}$, Timothy [Burks and Sakallah 1993] represented it by the following linear functions:

$$\begin{cases} x_{max} \geq x_1, \\ x_{max} \geq x_2, \\ x_{max} - x_1 \leq wg, \\ x_{max} - x_2 \leq (1-w)g \end{cases} \quad (15)$$

where, w is a binary variable, and g is a sufficiently large positive constant.

When $n \geq 3$, we can group them into pairs (if n is not even, let x_n itself be a group) and use Timothy's method to get the maximum value of each pair, then repeat on these maximum values iteratively until x_{max} is obtained. For example, when $n = 3$, we can change the format as $x_{max} = max\{max\{x_1, x_2\}, x_3\}$, and then represent each max operation using Timothy's method.

At this point we have addressed the issues outlined at the beginning of this section and have formulated the timing similarity problem to MBQP problem.

Commercial tools, such as IBM ILOG CPLEX which uses the branch and bound algorithm can considerably speed up the process of solving real-life applications with much lower average time complexity than the worst case [Thakoor and Gao 2011]. However MBQP is still NP-hard. Hence, when the number of faulty cores and backup cores are large, the execution time of IBM ILOG CPLEX could still be prohibitively long and hinders the MBQP based approach from being applicable for online reconfiguration. Hence a quicker solution is needed and the greedy-based algorithm is developed for this purpose. Next, we present the greedy algorithm which may not find the optimal solution, but can provide a good solution in polynomial time.

5.2. Greedy Approach

The MBQP approach is guaranteed to find the optimal solution, if one exists. The general idea of the greedy algorithm is to choose one defective core replacement at a time, making an optimal choice for each individual replacement, hoping to arrive at the optimal solution for the reconfiguration. Unfortunately, finding *the optimal* solution cannot be guaranteed in this manner as the choice of replacements are not independent and the order in which the defective cores are replaced has significant impact on the optimality of the result.

In our application model, tasks may have dependencies. In other words, changes in the starting time of precedent tasks may propagate to their dependent tasks. When selecting a replacement for a defective core $c_{(i)}$ which task v_i is deployed upon, the local optimal is to use a backup core that minimizes task v_i's starting time change. However, when task dependency has to be taken into consideration, minimizing the impact on the dependent tasks' starting time change becomes important. Hence, we choose to order the defective cores in descending order by the number of descendants of the tasks mapped to the core. This gives higher priority to cores which have the largest potential to affect other tasks.

As shown in the example in section 4.1, if a reconfiguration causes a link to exceed its bandwidth, it is not a valid configuration. This presents a problem to the greedy algorithm because it is possible to unknowingly choose replacements which leave only invalid options at a later point. In order to avoid this situation, we impose a precondition on the greedy algorithm, i.e., requiring that the sum of the communication rates on all edges incident to defective cores does not exceed the available bandwidth of any link in the NoC. More specifically, given a set of defective cores F, the precondition is defined by (16):

$$\sum_{e_j \in F^e} \frac{1}{rate(e_j)} \leq 1 - \max_{l_i \in L} \left(\sum_{e_k \in E - F^e} rate(e_k) \times M_{EL}^{ref}(k, i) \right) \tag{16}$$

where F^e is the set of edges in the task graph incident to a vertex which is mapped to a defective core:

$$F^e = \{e_i | e_i = (v_j, v_k), M_{\widetilde{C}C}^{ref}(M_{V\widetilde{C}}(v_j)) \in F \vee M_{\widetilde{C}C}^{ref}(M_{V\widetilde{C}}(v_k)) \in F\}$$

The satisfaction of condition (16) guarantees sufficient bandwidth on a link even when the communication over all the edges in F^e are moved to the link.

It is worth pointing out that the precondition is only a sufficient condition. In other words, when the precondition is not satisfied, the algorithm may still be able to find a solution using, for instance, backtracking. However, this will change the time complexity of the algorithm.

As mentioned before, the goal of the greedy algorithm is to quickly find a solution for on-line reconfiguration. It is not intended to overlap with MBQP. In addition, for a real-time application, a solution with guaranteed performance is critical. Hence, rather than using backtracking, we enforce the satisfaction of the precondition (16). The algorithm is given in Algorithm 1.

ALGORITHM 1: The Greedy Algorithm

```
Precondition: Formula (16)
sort $F$ descending by the number of descendants ;
$M^{new}_{\widetilde{C}C} \leftarrow M^{ref}_{\widetilde{C}C}$ ;
for $i \leftarrow 1$ to $|F|$ do
    $bs \leftarrow \infty$ ;
    $bc \leftarrow 0$ ;
    for $j \leftarrow 1$ to $|B|$ do
        $M^{new}_{\widetilde{C}C}(c_{(i)}) \leftarrow b_j$ ;
        if $D_{new} < bs$ then
            $bs \leftarrow D_{new}$ ;
            $bc \leftarrow j$;
        end
    end
    $M^{new}_{\widetilde{C}C}(c_{(i)}) \leftarrow b_{bc}$ ;
    $B \leftarrow B - \{b_{bc}\}$;
end
return $M^{new}_{\widetilde{C}C}$ ;
```

Defective cores which have not yet been remapped are assumed to be in their initial location in the reference mapping when calculating their tasks' starting times (line 2). For each defective core (line 3) the algorithm explores the possible backup cores as a replacement (lines 6-12), choosing the backup core which keeps timing distance minimized. Each time a backup core is chosen, it is removed from the list of choices for the next iteration (line 14). The greedy algorithm produces a new virtual topology mapping $M^{new}_{\widetilde{C}C}$ and set of backup cores B. The new mapping and remaining backup cores can be used as the reference mapping for future reconfigurations.

The time complexity of the greedy algorithm is $(|F| * |B|) \times (|V| + |E|) + |F| \lg |F|)$ where $|F| * |B|$ is the nested loop and $|V| + |E|$ is the time to calculate the timing similarity D_{new} (line 8). The time to sort the defective cores (line 1) is just $|F| \lg |F|$ as the number of descendants is static and can be stored with the task graph. However, as we pointed out earlier in the subsection that such polynomial time complexity is based on the assumption that the precondition, i.e., formula (16), holds, which can be restrictive for some applications. The performance of the greedy algorithm is evaluated in the next section.

5.3. Comparison with Previous Work

As we have now given our assumptions, problem definition and proposed solutions, we will briefly discuss our previous work on this topic appearing in [Yue et al. 2011]. There are two main differences in this paper from our previous work. One is the use of a guaranteed service NoC model. Previously when defining our timing similarity metric the main issue was quantifying congestion, as this was unpredictable and very dependent on implementation details of the application and NoC. We developed a metric called Traffic Flow Occupancy which attempted to keep the level of traffic over links equal before and after reconfiguration. However, as our concern was with real-time applications it was found that this metric was not well correlated with the actual timing properties of the application. Adopting the guaranteed service NoC model added an additional constraint to the problem but ultimately allowed us to accurately predict communication times and thus adopt the more useful metric of task starting times. Additionally, in our previous work, we used communication graphs instead of task graphs. In this context communication graphs are derived from task graphs by removing dependency information and only considering the average bandwidth required by each edge. Removing this simplification was also necessary in order to calculate tasks starting times.

In our previous work we also proposed a greedy algorithm, the main difference from the current algorithm being the order in which faulty cores are replaced. In our previous work we replaced faulty cores in order of decreasing Traffic Flow Occupancy. This strategy is also compared to our current method in Section 6.2.2.

6. EVALUATION

In this section, we set up multiple experiments for the following purposes:
1) Evaluate the metric given in formula (4) for computing the task's starting time by comparing it with the simulated starting time.
2) Investigate the execution time of IBM ILOG CPLEX for solving the formulated MBQP problem by increasing the number of defective cores, the number of backup cores and the number of tasks, and then evaluate the impact of the returned solution by limiting its execution time.
3) Compare the solution found by our proposed greedy algorithm with the optimal solution, RRCS, an average choice and our previous greedy algorithm, and then rank the greedy choice among the entire solution space.

In our experiments, we use TGFF [Dick et al. 1998] to generate randomized application task graphs and create a random mapping of the tasks to a 2D mesh NoC, which is simulated by NIRGAM [Jain et al. 2007]. The tasks to cores mapping is random because we do not assume anything about the original mapping of the application and only aim to achieve timing similarity.

We run IBM ILOG CPLEX on a Microsoft Windows XP server equipped with a dual-core E5200 Intel CPU, the clock speed of which is 2.5GHZ, and two gigabytes of memory to evaluate its execution time.

6.1. Metric Evaluation

In this set of experiments, we show that the metric given in formula (4) is a good model for predicting the task's starting time. We run 10 randomly generated applications on NIRGAM. The applications contain 135 tasks in all and are randomly mapped to a 6×6 mesh NoC. For each task in the applications the starting time is recorded from the results of NIRGAM and calculated using formula (4), the delta is taken and normalized to the calculated started time:

$$normalized\Delta(v_i) = \frac{|sStart^{ref}(v_i) - start^{ref}(v_i)|}{start^{ref}(v_i)} \tag{17}$$

where $sStart^{ref}(v_i)$ is the simulated starting time of v_i on the reference mapping given by the NIRGAM simulation.

The results of the experiments are shown in Fig. 2. The average value of these normalized starting time differences is 1.24%, and the standard deviation is 0.7%. Based on these results, we can conclude that our model is sufficiently accurate to predict tasks' starting times compared to the NIRGAM simulator, which takes into account the low level details of the NoC.

6.2. Algorithm Evaluation

6.2.1. MBQP Approach.

Three sets of experiments are designed in this section. The first two are to investigate how the number of defective cores, the number of backup cores and the number of tasks impact the running time of IBM ILOG CPLEX to solve the formulated MBQP problem.

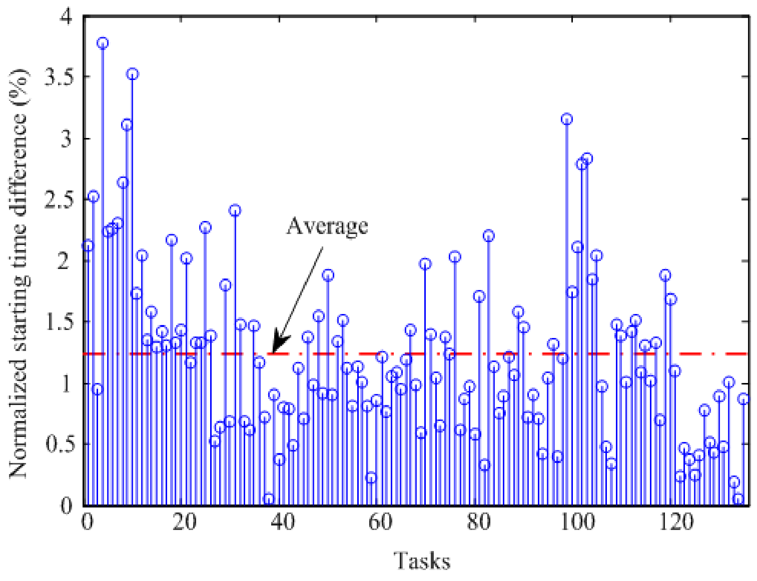

Fig. 2. Model accuracy simulation

The third is to evaluate the impact to the solution returned by limiting the running time of IBM ILOG CPLEX. The combinations of the backup cores and defective cores are randomly selected.

In the first set, we run an application with 16 tasks on a 6×6 mesh NoC. We set the number of backup cores as 8 and increase the number of defective cores from 2 to 8. For each scenario, we get the average time cost by running 10 cases, each with randomly selected backup and defective core sets. We then increase the number of backup cores to 12 and 20 and repeat the experiments. The results are shown in Fig. 3. We can see that the running time increases more quickly when the number of backup cores increases. Note the logarithmic scale of the graph.

The second experiment is designed to evaluate the impact of the number of tasks on the running time. The number of defective and backup cores are fixed at 6 and 12 respectively with a mesh size of 8×8. We then generate applications containing between 8 and 48 tasks. Ten combinations of backup and defective cores are chosen for each task size and the average running time is graphed in Fig. 4. We can see the time cost increases from 7 seconds to 21 seconds when the number of tasks grows from 8 to 48. Comparing with Fig. 3, we can see the running time is affected more by the number of defective cores and the number of backup cores than the number of tasks, the reason is the first two determine the search space of the solution.

The last test is based on the case with 8 defective cores and 20 backup cores in the first set of experiments, which costs about 1760 seconds to obtain the optimal solution with a timing distance of 7. As shown in Fig. 5, by limiting the execution time to 10, 100 and 600 seconds, the solutions returned by IBM ILOG CPLEX have the timing distance of 29, 22 and 19, respectively.

From these experiments, we have the following observations:

1) The running time of IBM ILOG CPLEX is affected more by the number of defective cores and the number of backup cores than the number of tasks.

2) By limiting the execution time, IBM ILOG CPLEX can return an approximate solution, and the longer the execution time, the better the solution.

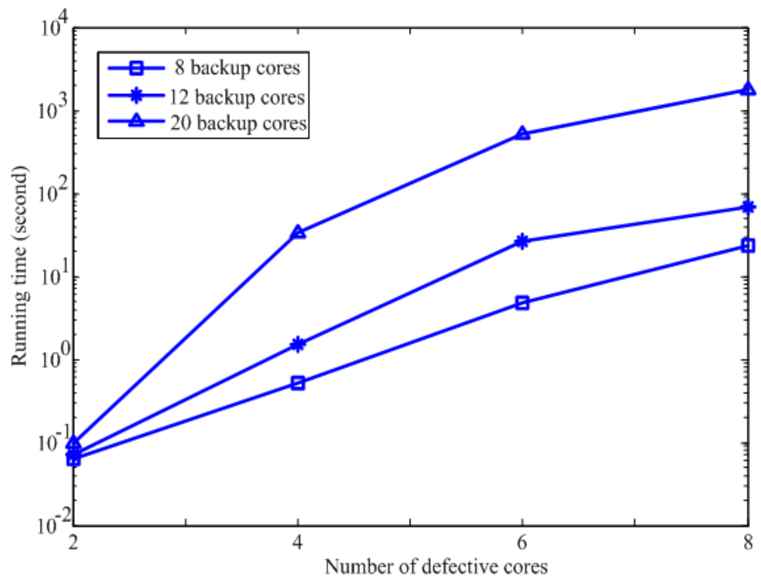

Fig. 3. Running time impact corresponding to the number of defective cores and number of backup cores

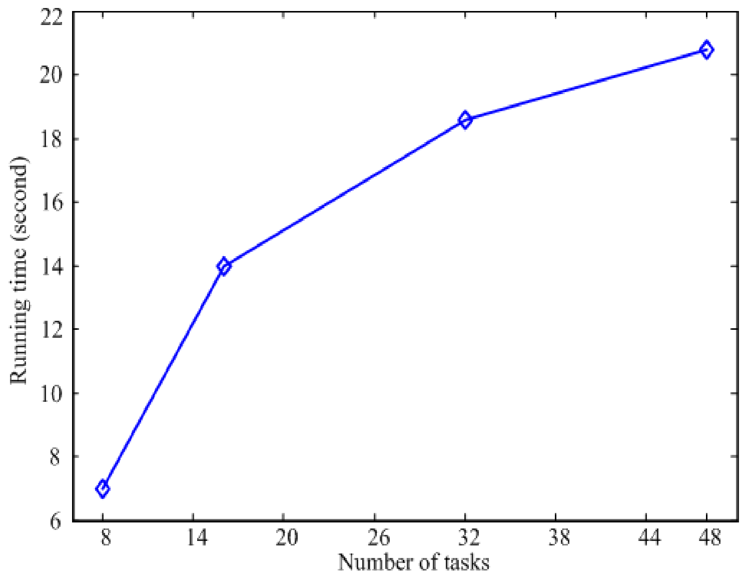

Fig. 4. Running time impact corresponding to the number of tasks

6.2.2. Greedy Approach.

In this section, we evaluate the performance of the proposed greedy algorithm. We first compare the timing similarity of the solution found by the greedy algorithm with the optimal one, RRCS [Zhang et al. 2008], our previous greedy algorithm [Yue et al. 2011] and a randomly selected solution. We then examine how the algorithms effect the starting times of a real application. Finally we explore the whole solution space and compare the greedy choice among them.

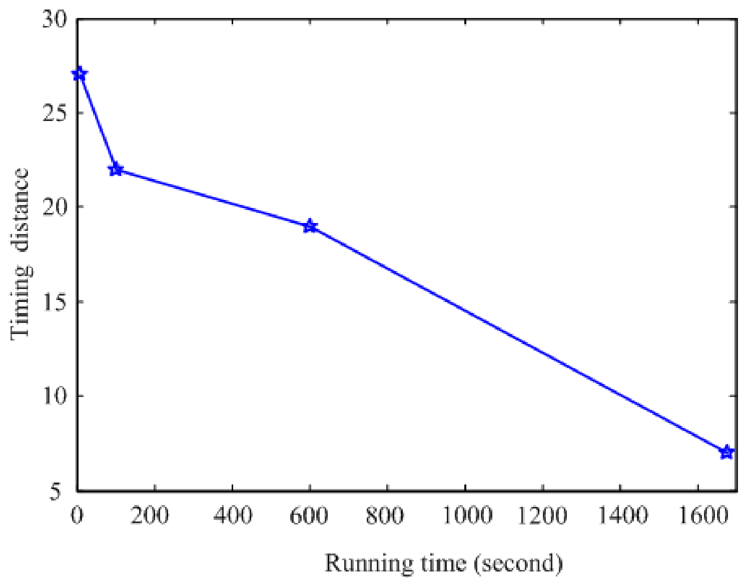

Fig. 5. Impact of the solution by limiting the running time

The experiment is put in the context of mesh sizes to easily relate results back to the NoC. Each mesh contains an $n \times n$ virtual topology with a $(n+1) \times n$ physical topology. For each mesh size, 15 random applications are generated using TGFF [Dick et al. 1998] and are randomly mapped to the virtual topology. The size of the task graph is $n \times n$. For each task graph, we generate 15 random faulty core sets using the maximum number of redundant cores available, i.e., 6 faulty cores for a 6×6 mesh. We then solve each problem instance using each algorithm and average the resulting timing similarity across all 225 cases. The random choice can be understood as an average timing similarity for its respective mesh size.

Fig. 6 gives the results of the first experiment. In general, we see the greedy algorithm is closer to the optimal, on average, than the other algorithms. Looking to the RRCS algorithm we see that it performs slightly better than the optimal at the 3×3 mesh size. This is because RRCS is not constrained to only remapping defective cores as our other algorithms are. Note that as the mesh size grows, RRCS quickly loses its advantage as it only considers the location of the faulty cores and is agnostic to the application.

Looking at our previous greedy algorithm, we see that the solution becomes increasingly poor compared to the current greedy algorithm as the size of the problem increases. This is due to the previous greedy algorithm's order of replacement. The larger applications will see increasingly poor performance as increased starting time of precedent tasks propagates to successors.

Now we will take a real application from the Embedded Systems Synthesis benchmark (E3S) suite and examine how the starting time of individual task varies. The E3S also uses task graphs to specify applications, although it is not a real substitute for real applications, this synthetic benchmark models realistic application data using real-world applications and systems [Dick 2007]. The application we chosen is task graph 2 from the Auto Industry suite, which is the largest one in the suite. The application contains nine tasks and is mapped to a 4×4 virtual topology supported by a 5×4 physical topology. We randomly pick 4 defective cores. The timing similarity of the

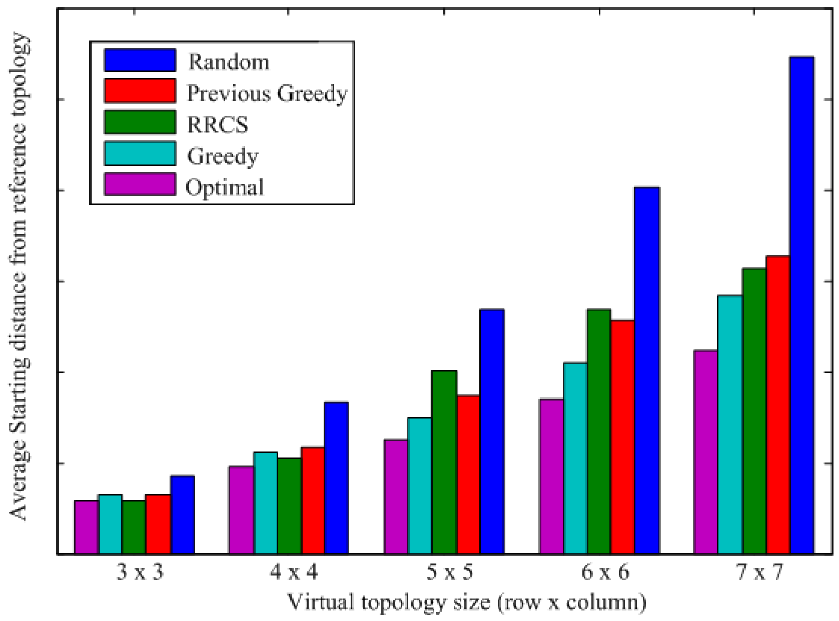

Fig. 6. Comparing greedy choice with other algorithms

solutions given by RRCS, our previous greedy algorithm, our current greedy algorithm and the optimal solution are 5.0, 2.6, 2.4 and 2.0 respectively. We also give the average starting time delta by enumerating the entire solution space. The tasks are ordered along the x-axis by the number of descendants they contain in the graph. The results are shown in Fig. 7.

For the final experiment we will examine how the greedy algorithm ranks among the entire solution space. We consider several combinations of mesh, backup and defective core sizes. We generate 25 problems for each combination. For each problem we sort the solution space by timing similarity and record the position of the greedy solution as a percentile rank. The results are given in Table II and include the worst, average and standard deviation among the 25 problems for each size.

From Table II, we have the following observations:

1) The performance of the greedy algorithm is stable, the average case of the greedy choices is among the top 10% of all the possible solutions.

2) When the mesh size becomes larger, the average cases and most of the worst cases get better, the standard deviations also become smaller. This is due to the exponentially increasing size of the solution space containing a larger number of poor choices for reconfiguration.

7. CONCLUSION

Virtualization in many-core systems in presence of manufacturing defects and device wear-outs for real-time and multimedia applications is very complex and depends heavily on the desired hardware architecture, timing requirements, and the on-chip backup core distributions. The developed MBQP approach is able to find the optimal candidates to replace defective cores for the reconfiguration, so that the timing behaviors to replace the mapped application is most similar to the one initially designed. However, the MBQP approach is time consuming, especially when the number of backup cores and the number of defective cores are large. Hence, this approach is

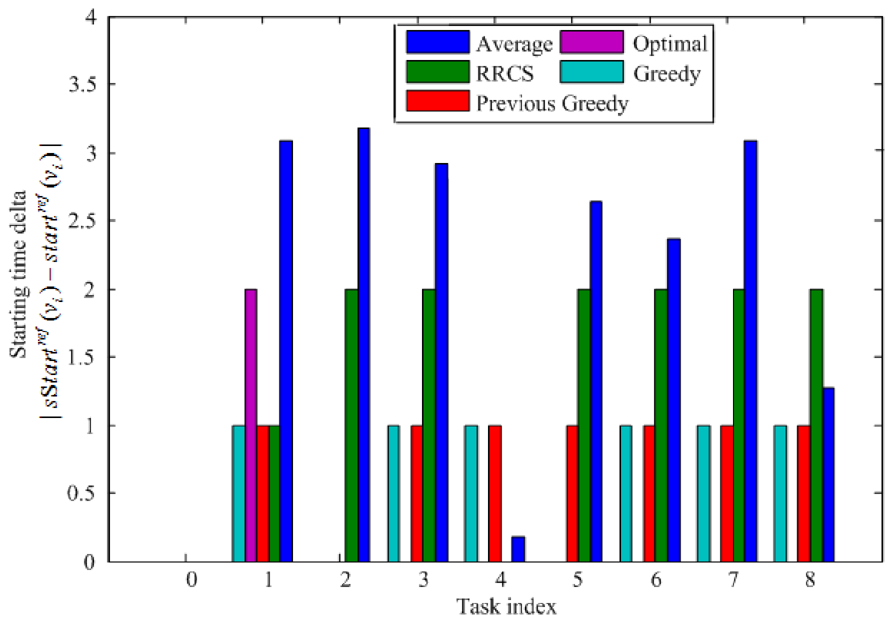

Fig. 7. Algorithms comparison in term of task's starting time

Table II Ranks of the greedy choice

mesh size	tasks	backup cores	defective cores	worst (%)	average (%)	standard deviation (%)
4×4	11	5	3	45.84	5.74	13.51
			4	75.00	8.54	16.20
			5	47.67	9.98	14.68
5×5	20	5	3	56.67	4.60	13.45
			4	40.84	4.19	12.41
			5	43.20	3.29	12.02
6×6	30	6	3	30.00	2.08	11.03
			4	32.50	2.26	11.44
			5	24.65	2.54	11.21

more suitable for offline reconfiguration. In order to guarantee to find a good solution in short time, a more efficient heuristic algorithm, i.e., greedy algorithm is proposed, which can be used for the online reconfiguration. The simulation results show that the average case of the solutions obtained by the greedy approach are among the top 10% of all the possible ones and improve upon previous methods.

The research presented in the paper is only the first step toward applying virtualization technologies to real-time and multimedia applications. We are all aware that the precondition imposed on the heuristic algorithm may not always be established, our immediate next step is to relax this condition to cover more scenarios. The developed MBQP approach is time consuming and may only be acceptable for offline reconfiguration for now, but we plan to use rounding techniques to transform this problem to a quadratic programming (QP) problem, which can be solved in polynomial time.

We will study how reconfiguration may impact hard real-time applications, and investigate virtualization techniques that guarantee deadline satisfactions. In addition to the timing issues, real-time and multimedia 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. Another direction of our future work is to address these concerns for heterogeneous many-core systems.

REFERENCES

Daniel Axehill. 2005. *Applications of Integer Quadratic Programming in Control and Communication*. Ph.D. Dissertation. Department of Electrical Engineering Köping University.

L. Benini and G. De Micheli. 2002. Networks on chips: a new SoC paradigm. *IEEE Computers* 35, 1 (Jan. 2002), 70–78.

T. Bjerregaard and J. Sparso. 2005. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proc. of IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*. 1226–1231.

T.M. Burks and K. Sakallah. 1993. Min-max linear programming and the timing analysis of digital circuits. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 152–155.

Chen-Ling Chou and R. Marculescu. 2008. Contention-aware application mapping for Network-on-Chip communication architectures. In *Proc. of IEEE International Conference on Computer Design (ICCD)*. 164–169.

W. J Dally and B. Towles. 2001. Route packets, not wires: on-chip interconnection networks. In *Proc. of 38th Design Automation Conference (DAC)*. 684–689.

O. Derin, D. Kabakci, and L. Fiorin. 2011. Online task remapping strategies for fault-tolerant Network-on-Chip multiprocessors. In *Proc. of 5th IEEE/ACM International Symposium on Networks on Chip (NoCS)*. 129–136.

R. Dick. 2007. Embedded System Synthesis Benchmarks (E3S). (2007). Available from http://ziyang.eecs.umich.edu/ dickrp/e3s/.

Robert P. Dick, David L. Rhodes, and Wayne Wolf. 1998. TGFF: task graphs for free. In *Proc. of the 6th international workshop on Hardware/software codesign (CODES/CASHE)*. 97–101.

K. Goossens, J. Dielissen, and A. Radulescu. 2005. AEthereal network on chip: concepts, architectures, and implementations. *IEEE Design Test of Computers* 22, 5 (2005), 414–421.

Gu Haiyun, Li Changwen, and Sun Shu. 2007. Research on mapping algorithm of irregular mesh NoC for portable multimedia appliances. In *IET Conference on Wireless, Mobile and Sensor Networks (CCWMSN)*. 697–700.

T. Hastie, R. Tibshirani, and J.H. Friedman. 2008. *The elements of statistical learning: data mining, inference, and prediction*. New York: Springer-Verlag.

Tom Henzinger, Rupak Majumdar, and Vinayak Prabhu. 2005. Quantifying similarities between timed systems.. In *Proc. of the 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 226–241.

IBM. 2008. IBM ILOG CPLEX Optimizer. (Nov. 2008). http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

L. Jain, BM Al-Hashimi, MS Gaur, V. Laxmi, and A. Narayanan. 2007. NIRGAM: a simulator for NoC interconnect routing and application modeling. In *Workshop on Diagnostic Services in Network-on-Chips, Design, Automation and Test in Europe Conference (DATE)*. 16–20.

A. Lankes, A. Herkersdorf, S. Sonntag, and H. Reinig. 2009. NoC topology exploration for mobile multimedia applications. In *Proc. of 16th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. 707 –710.

Tang Lei and S. Kumar. 2003. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proc. of IEEE Euromicro Symposium on Digital System Design*. 180 –187.

Jane W. S. W. Liu. 2000. *Real-Time Systems* (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

Ning Ma, Zhonghai Lu, Zhibo Pang, and Lirong Zheng. 2010. System-level exploration of mesh-based NoC architectures for multimedia applications. In *Proc. of IEEE International SOC Conference (SOCC)*. 99–104.

Ed Sperling. 2007. Turn down the heat...please. (2007). Available from http://dopu.cs.auc.dk.

Ninad Thakoor and Jean Gao. 2011. Branch-and-Bound for Model Selection and Its Computational Complexity. *IEEE Trans. on Knowledge and Data Engineering* 23, 5 (May 2011), 655–668.

L.A. Wolsey. 1998. *Integer programming*. Wiley-Interscience, NY, USA.

Y. Yu, S. Ren, and O. Frieder. 2010. Feasibility of semiring-based timing constraints. *ACM Trans. on Embedded Computing Systems* 9, 4 (Sep. 2010), 33.

Ke Yue, S. Ghalim, Zheng Li, F. Lockom, Shangping Ren, Lei Zhang, and Xiaowei Li. 2011. A greedy approach to tolerate defect cores for multimedia applications. In *Proc. of 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*. 112–119.

Lei Zhang, Yinhe Han, Qiang Xu, and Xiaowei Li. 2008. Defect Tolerance in Homogeneous Manycore Processors Using Core-Level Redundancy with Unified Topology. In *Proc. of IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*. 891–896.

Lei Zhang, Yinhe Han, Qiang Xu, Xiao wei Li, and Huawei Li. 2009. On Topology Reconfiguration for Defect-Tolerant NoC-Based Homogeneous Manycore Systems. *IEEE Trans. on Very Large Scale Integration Systems* 17, 9 (Sep. 2009), 1173–1186.

Lei Zhang, Yue Yu, Jianbo Dong, Yinhe Han, Shangping Ren, and Xiaowei Li. 2010. Performance-asymmetry-aware topology virtualization for defect-tolerant NoC-based many-core processors. In *Proc. of IEEE/ACM Design, Automation Test in Europe Conference (DATE)*. 1566–1571.

Received January 2011; revised July 2012, January 2013; accepted May 2013