1-2013

Framework for Simulation of Heterogeneous MpSoC for Design Space Exploration

Bisrat Tafesse
University of Nevada, Las Vegas

Venkatesan Muthukumar
University of Nevada, Las Vegas, vm@unlv.nevada.edu

Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 936181, 16 pages
http://dx.doi.org/10.1155/2013/936181

Research Article

Framework for Simulation of Heterogeneous MpSoC for Design Space Exploration

Bisrat Tafesse and Venkatesan Muthukumar

Department of Electrical and Computer Engineering, University of Nevada Las Vegas, 4505 Maryland Pkwy, Las Vegas, NV 89154, USA

Correspondence should be addressed to Venkatesan Muthukumar; venkatesan.muthukumar@unlv.edu

Received 8 October 2012; Revised 29 January 2013; Accepted 14 April 2013

Academic Editor: Paul Bogdan

Due to the ever-growing requirements in high performance data computation, multiprocessor systems have been proposed to solve the bottlenecks in uniprocessor systems. Developing efficient multiprocessor systems requires effective exploration of design choices like application scheduling, mapping, and architecture design. Also, fault tolerance in multiprocessors needs to be addressed. With the advent of nanometer-process technology for chip manufacturing, realization of multiprocessors on SoC (MpSoC) is an active field of research. Developing efficient low power, fault-tolerant task scheduling, and mapping techniques for MpSoCs require optimized algorithms that consider the various scenarios inherent in multiprocessor environments. Therefore there exists a need to develop a simulation framework to explore and evaluate new algorithms on multiprocessor systems. This work proposes a modular framework for the exploration and evaluation of various design algorithms for MpSoC system. This work also proposes new multiprocessor task scheduling and mapping algorithms for MpSoCs. These algorithms are evaluated using the developed simulation framework. The paper also proposes a dynamic fault-tolerant (FT) scheduling and mapping algorithm for robust application processing. The proposed algorithms consider optimizing the power as one of the design constraints. The framework for a heterogeneous multiprocessor simulation was developed using SystemC/C++ language. Various design variations were implemented and evaluated using standard task graphs. Performance evaluation metrics are evaluated and discussed for various design scenarios.

1. Introduction

Current uniprocessor systems guarantee high processor utilization; however, they fall short in meeting critical real time demands that require (1) high data throughput, (2) fast processing time, (3) low energy and power consumptions, and (4) fault tolerance. Present-day applications that run on embedded or uniprocessor systems, such as multimedia, digital signal processing, image processing, and wireless communication, require an ever-growing amount of resources and speed for data processing and storage, which can no longer be satisfied by uniprocessor systems [1]. The increasing performance demand dictates the need for concurrent task processing on multiprocessor systems. Multiprocessor SoC (MpSoC) systems have emerged as a viable alternate to address the bottlenecks of uniprocessor systems. In an effort to improve the processing performance, multiprocessor systems incorporate a number of processors or processing elements (PEs). A typical MpSoC system is composed of the following major components:

(1) multiple processing elements (PEs) that perform the task execution,

(2) multiple memory elements (MEs) that are used for data storage, and

(3) a communication network that interconnects the processors and memories in a manner defined by the topology.

Heterogeneity in MpSoCs [2] is described as the variation of the functionality and flexibility of the processor or the

memory element. Heterogeneous processors in MpSoCs can be programmable or dedicated (nonprogrammable) processors. Also, memory elements can be heterogeneous with respect to their memory type, size, access mode, and clock cycles. To transfer the data to and from the processing and memory elements, the network component may include busses, switches, crossbars, and routers.

Present-day MpSoC architectures employ most of the above discussed components [3]. In fact, to effectively meet today's complex design requirements, modern multiprocessor systems are real time, preemptive, and heterogeneous [4]. For data exchange, memory elements in MpSoCs often follow either distributed memory architecture (message passing) or a global shared memory architecture [5–7].

An efficient MpSoC is based upon several key design choices, also known as design space exploration: network topology selection, good routing policy, efficient application scheduling, mapping, and the overall framework implementation. A formal categorization of MpSoC design issues presented in [8] is summarized below. Design methodologies in multiprocessor systems (MpSoC and NoC) have been classified as follows: (1) application characterization, scheduling, and mapping; (2) processing, memory, switching and communication architecture modeling, development, and analysis; and (3) simulation, synthesis, and emulation framework.

Major design features that need to be addressed for multiprocessors are task scheduling, mapping, and fault tolerance. This work concentrates on developing a simulation framework for multiprocessors on a single chip (MpSoC) that simplifies design space exploration by offering comprehensive simulation support. The simulation framework allows the user to optimize the system design by evaluating multiple design choices in scheduling, mapping, and fault tolerance for the given design constraints. TILE and TORUS topologies, which are common layouts in multiprocessing environments, are supported in this framework.

Using the developed MpSoC simulation framework, we propose and evaluate (1) an efficient strategy for task scheduling, called the performance driven scheduling (PDS) algorithm based on Simulated Annealing heuristic that determines an optimal schedule for a given application; (2) a new mapping policy called Homogenous Workload Distribution (HWD), which maps tasks to processors by considering processors runtime status; and (3) finally, in an effort to implement a robust system, a simple fault-tolerant (FT) algorithm that reconfigures task execution sequence of the system in the event of processor failure. Heterogeneity in processor architecture is a key design feature in multiprocessor systems and therefore is modeled in this framework. Even though many earlier multiprocessor designs solve the scheduling and mapping problem together, our framework considers them individually for fault-tolerance.

In summary, the research presented in this paper addresses the following issues and solutions.

(1) Develop a comprehensive MpSoC framework implemented using C++ and SystemC programming languages for effective exploration of design choices like application scheduling, mapping, and architecture design.

(2) The developed system can also evaluate MpSoC system architecture design choices differing in the number and type of processing elements (PEs) and topology.

(3) The paper proposes a heuristic scheduling algorithm based on Simulated Annealing.

(4) Also, a mapping algorithm that distributes the workload among the available PE is proposed.

(5) Finally, a centralized fault-tolerant scheme to handle faults in PEs is proposed.

This paper is organized as follows. Section 2 briefly summarizes previous works in the field of multiprocessor systems. Section 3 provides definitions and terminologies used in this work. Section 4 presents the framework description as well as the proposed algorithms and their implementation in task scheduling, mapping, and fault tolerance techniques. Section 5 discusses the experimental setup for simulation and evaluations. Section 6 discusses the detailed simulation results and the analysis of the algorithms on benchmarks. Finally, Section 7 summarizes the conclusions.

2. MpSoC Design Literature Review

Extensive work has been done in the field of MpSoC- and NoC-based systems. Earlier research targeted a variety of solutions including processor architecture, router design, routing and switching algorithms, scheduling, mapping, fault tolerance, and application execution in multiprocessors. to address the performance and area demands. The readers can refer to [8–10] for an extensive enumeration of NoC research problems and proposed solutions. In this section, we concentrate on the earlier proposed scheduling, mapping, and fault-tolerant algorithms. It has been shown that earlier scheduling algorithms like FCFS and Least Laxity [7] fail to qualify as acceptable choices of scheduling because they are unable to meet deadline (hard deadlines) requirements for real-time applications. Thai [7] proposed Earliest Effective Deadline First (EEDF) scheduling algorithm. This work demonstrated that increasing the number of processors to improve the scheduling success increased the size of the architecture, which makes task scheduling more difficult. Stuijk et al. [11] proposed dynamic scheduling and routing strategies to increase resource utilization by exploiting all the scheduling freedom on the multiprocessors. This work considered application dynamism to schedule applications on multiprocessors both for regular and irregular tiled topology. Hu and Marculescu [12] propose an algorithm based on list scheduling where mapping is performed to optimize energy consumption and performance. Varatkar and Marculescu [13] propose an ILP-based scheduling algorithm that minimizes interprocess communication in NoCs. Zhang et al. [14] propose approximate algorithms for power minimization of EDF and RM schedules using voltage and frequency scaling. Also other researchers [15–17] have used dynamic

voltage scaling in conjunction with scheduling for low power requirements.

Finding an optimal solution in scheduling and mapping problem domain for set of real-time tasks has been shown to be NP-hard problem [11]. Carvalho et al. [18], proposed a "congestion-aware" mapping heuristic to dynamically map tasks on heterogeneous MpSoCs. This heuristic approach initially assigns master-tasks to selected processors and the slave-tasks are allocated on congestion minimizing manner to reduce the communication traffic. Sesame project [4], proposed a multiprocessor model that maps applications together with the communication channels on the architecture. This model tries to minimize an objective function, which considers power consumption, application execution time and total cost of the architectural design. Many researchers have discussed static and dynamic task allocation for deterministic and non-deterministic tasks extensively [5]. Static mapping has also been shown non-efficient strategy because it does not take application dynamism into account that is a core design issues in multiprocessor systems. Ahn et al. [19], suggests that real-time applications should behave deterministically even in unpredictable environments so that the tasks can be processed in real-time. Ostler and Chatha [20] propose an ILP formulation for application mapping on network processors. Harmanani and Farah [21], propose an efficient mapping and routing algorithm that minimizes blocking and increases bandwidth based on simulated annealing. Also mapping algorithms to minimize communication cost, energy, bandwidth and latency either individually [22–24] or as a multi-objective criteria [25, 26] have been proposed.

Task migration as opposed to dynamic mapping has been proposed in [4] to overcome performance bottlenecks. Task migration may result in performance improvement in shared memory systems but may not be feasible when considered for distributed memory architectures due to the high volume of inter-processor data communication [5].

Previous research in [5, 6, 27] has demonstrated reliability issues regarding application processing, data routing and communication link failure. Zhu and Qin [28], demonstrated a system level design framework for a tiled architecture to construct a reliable computing platform from potentially unreliable chip resources. This work proposed a fault model, where faults may have transient and permanent behaviour. Transient errors are monitored by a dedicated built-in checker processor that effectively corrects the fault at runtime. Nurmi et al. [6] demonstrated error tolerance methods through time, space and data redundancy. This is however a classic and costly approach with respect to system resources and bandwidth though the work demonstrated redundancy is one potential approach to address faulty scenarios. Holsmark et al. [29], proposed a fault tolerant deadlock free algorithm (APSRA) for heterogeneous multiprocessors. APSRA stores routing information in memory to monitor faulty routs and allow re-configurability dynamically. Pirretti et al. [30] and Bertozzi et al. [31] propose NoC routing algorithms that can route tasks around the fault and sustain network functionality. Also research on router design to improve reliability [32], buffer flow control [33] and error correction techniques [34, 35] for NoC have been proposed. Koibuchi et al. [36] proposed a lightweight fault-tolerant mechanism called default backup path (DBP). The approach proposed a solution for non-faulty routers and healthy PEs. They design NoC routers with special buffers and switch matrix elements to re-route packets through special paths. Also, Bogdan et al. [37] propose an on-chip stochastic communication paradigm based on probabilistic broadcasting scheme that takes advantage of the large bandwidth available on the chip. In their method, messages/packets are diffused across the network using spreaders and are guaranteed to be received by destination tiles/processors.

A number of EDA research groups are studying different aspects of MpSoC design, some of which include the following. (1) Polis [18] is a framework based on both microcontrollers and ASICs for software and hardware implementation. (2) Metropolis [18] is an extension of Polis that proposed a unified modeling and structure for simulating computation models. (3) Ptolemy [18] is a flexible framework for simulating and prototyping heterogeneous systems. (4) Thiele et al. [1] proposed a simulation based on distributed operation layer (DOL) which considers concurrency, scalability, mapping optimization, and performance analysis in an effort to yield faster simulation time. (5) NetChip: xpipes, xpipes compiler, and SUNMAP. NetChip is a NoC synthesis environment primarily composed of two tools, namely, SUNMAP [38] and the xpipes compiler [39]. (6) NNSE: Nostrum NoC Simulation Environment. NNSE [40] is a SystemC-based NoC simulation environment initially used for the Nostrum NoC. (7) ARTS Modeling Framework: ARTS [41] is a system-level framework to model networked multiprocessor systems on chip (MpSoC) and evaluate the cross-layer causality between the application, the operating system (OS), and the platform architecture. (8) StepNP: a System-Level Exploration Platform for Network Processors. StepNP [42] is a System-Level Exploration Platform for Network Processing built in SystemC. It enables the creation of multiprocessor architectures with models of interconnects (functional channels, NoCs), processors (simple RISC), memories, and coprocessors. (9) Genko et al. [43] present a flexible emulation environment implemented on an FPGA that is suitable to explore, evaluate, and compare a wide range of NoC solutions with a very limited effort. (10) Coppola et al. [44] proposes an efficient, open-source research and development framework (On-Chip Communication Network (OCCN)) for the specification, modeling, and simulation of on-chip communication architectures. (11) Zeferino and Susin [45] present SoCIN, a scalable network based on a parametric router architecture to be used in the synthesis of customized low cost NoCs. The architecture of SoCIN and its router are described, and some synthesis results are presented.

The following papers presented are the closest works related to the proposed algorithms discussed in this paper. Orsila et al. [46] investigated the use of Simulated Annealing-based mapping algorithm to optimize energy and performance on heterogeneous multiprocessor architecture. In specific the cost function considered for SA optimization

includes processor area, frequency of operation, and switching activity factor. The work compares the performance of their SA-mapping algorithms for different heterogeneous architectures. In another similar work [47], proposed by the same author, the SA-mapping algorithm is adopted to optimize usage of on-chip memory buffers. Their work employs a B-level scheduler after the mapping process to schedule tasks on each processor.

The proposed framework discussed in this paper employs SA-based scheduling and workload balanced mapping algorithms. Our algorithms also optimize scheduling and mapping cost functions that include processor utilization, throughput, buffer utilization, and port traffic along with processor power. In addition, our algorithms also consider task deadline constraints. Both scheduling and mapping algorithms adopted in our work perform global optimization.

Paterna et al. [48] propose two workload allocation/mapping algorithms for energy minimization. The algorithms are based on integer linear programming and a two-stage-heuristic mapping algorithm based on linear programming and bin packing policies. Teodorescu and Torrellas [49] and Lide et al. [50, 51] explore workload-based mapping algorithms for independent tasks, and dependent tasks respectively. The work by Zhang et al. is based on ILP optimization for execution time and energy minimization. Our proposed mapping algorithm varies from the above-specified workload constrained algorithms by considering the buffer size usage and execution time of the tasks in the buffer.

Works by Ababei and Katti [52] and Derin et al. [53] have used remapping strategies for NoC fault tolerance. In the solution proposed by Ababei and Katti [52], fault tolerance in NoCs is achieved by adaptive remapping. They consider single and multiple processor (PE) failures. The remapping algorithm optimizes the remap energy and communication costs. They perform comparisons with a SA-based remapper. Orsila et al. [46] propose an online remapping strategy called local nonidentical multiprocessor scheduling heuristic (LNMS) based on integer linear programming (ILP). The ILP minimizing cost functions include communication and execution time. The proposed NoC fault-tolerant algorithm employs rescheduling and remapping strategies to optimize processor performance and communication cost. Also task deadline constraints are considered during the rescheduling and remapping.

With the current design trends moving towards multiprocessor systems for high performance and embedded system applications, the need to develop comprehensive design space exploration techniques has gained importance. Many research contributions in development of hardware modules and memory synchronization to optimize power, performance, and area for multiprocessor systems are being developed. However, algorithmic development for multiprocessor systems like multiprocessor scheduling and mapping has yielded significant results. Therefore, a comprehensive and generic framework for a multi-processor system is required to evaluate the algorithmic contributions. This work addresses the above requirement in developing a comprehensive MpSoC evaluation framework for evaluating many scheduling, mapping, and fault-tolerant algorithms. Also, the framework is developed on generic modeling or representation of applications (task graphs), architecture, processors, topologies, communication costs, and so forth resulting in a comprehensive design exploration environment.

The main contributions of this work can be summarized as follows.

(1) We propose a unique scheduling algorithm for MpSoC system called performance driven scheduling algorithm that is based on Simulated Annealing and optimizes an array of performance metrics such as task execution time, processor utilization, processor throughput, buffer utilization, and port traffic and power. The proposed algorithm performance is compared to that of the classical earliest deadline first scheduling algorithm.

(2) This paper proposes a mapping algorithm that succeeded the scheduling algorithm called the homogeneous workload distribution algorithm that distributes tasks evenly to the available processors in an effort to balance the dynamic workload throughout the processors. The proposed algorithm performance is compared with other classical mapping algorithm.

(3) Also, a fault-tolerant scheme that is effective during PE failure is proposed in this paper. The proposed fault-tolerant (FT) scheme performs an updated scheduling and mapping of the tasks assigning to the failed PE when the PE fault is detected. The unique contribution of this work is the fact that it addresses the pending and executing tasks in the faulty PEs. The latency overhead of the proposed FT algorithm is evaluated by experimentation.

3. Definitions and Theory

Simulation Model. The simulation model consists of three submodels (Figure 1): (1) application, (2) architecture, and (3) topology. The application model represents the application to be executed on the given MpSoC architecture and connected by the topology specified.

Application Model (Task Graphs). A task graph is a directed acyclic graph (DAG) that represents an application. A task graph is denoted as TG, such that TG = (V, E), where V is a set of nodes and E is a set of edges. A node in V represents a task such that $V = \{T_1, T_2, T_3, \ldots, T_N\}$, and an edge in E represents the communication dependency between the tasks such that $E = \{C_1, C_2, C_3, \ldots, C_K\}$. A weighted edge, if specified, denotes the communication cost incurred to transfer data from a source task to destination task. Each task can exist in a processor in one of the following states: ideal, pending, ready, active, running, or complete. The task graph is derived by parsing the high-level description of an application.

Architecture Model. The architecture of the MpSoC system consists of the processors, hardware components,

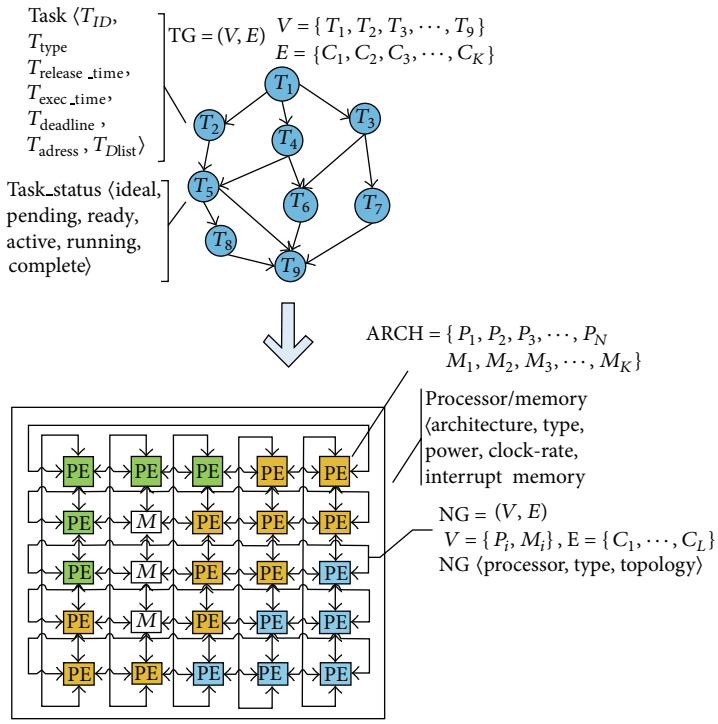

Figure 1: Simulation model containing application, architecture, and topology.

memory, and interconnects as defined by the topology model. The MpSoC architecture is represented as a set, $\{P_1, P_2, P_3, \ldots, P_N, M_1, M_2, M_3, \ldots, M_K\}$, where $P_1, P_2, \ldots, P_n$ are a set of N processors and $M_1, M_2, \ldots, M_K$ are a set of K memory units. Each processor P_i or memory M_i is defined by the following characteristics: type, power/task, execution/task, clock rate, preempt overhead, and also by a set of tasks $\{T_a, T_b, T_c, \ldots, T_z\}$ each processor can execute, along with their respective execution times and power costs. The implemented heterogeneous MpSoC consists of processors of the same architecture but can execute various set of tasks.

Topology Model (Topology Graph). Topology describes the physical layout of hardware components in a system. The topology model is represented as a topology graph (NG) that consists of the set of processors, topology type ⟨tile, torus⟩, and communication cost. This communication cost is the same communication cost (E) as defined in the application model.

Scheduling. Scheduling is the process of assigning an execution start time instance to all tasks in an application such that tasks are executed within their respective deadline. For an application $A = \{T_1, T_2 \cdots T_N\}$, where T_i is a task instance in the application, task scheduling, denoted as $\alpha(T_i)$, is defined as an assignment of execution start time (t_i) for the task T_i in the time domain such that

$$\alpha(T_i) = t_i, \qquad t_i + e_i \leq d_i, \tag{1}$$

where e_i is the worst-case execution time of the task on a specific processor, d_i is the deadline of the task T_i, and t_i is the execution start time of task T_i. Scheduling of an application A is denoted as

$$\alpha(A) = \{\alpha(T_1), \alpha(T_2), \alpha(T_3) \cdots \alpha(T_N)\}. \tag{2}$$

If there exists a precedence operation between any two tasks such that the execution of a task T_i should occur before the execution of task T_j, represented as $T_i < T_j$, then:

$$t_i + e_i \leq t_j \leq d_i, \qquad t_j + e_j \leq d_j, \tag{3}$$

where t_i is the start time of task T_j, and e_j and d_j are the execution time and deadline of task T_j, respectively.

Performance Index (PI). PI is the cumulative cost function used to determine the optimal schedule by using the Simulated Annealing technique. PI quantifies the performance of the system, taking into consideration the averaged values of processor execution time (ETIME), utilization (UTL), throughput (THR), buffer utilization/usage (BFR), port traffic (PRT), and power (PWR). PI is evaluated by a cost function that is expressed as

$$\begin{aligned} \text{COST} = {} & \alpha_1 \left(\frac{1}{\text{ETIME}}\right) + \alpha_2 \left(\frac{1}{\text{UTL}}\right) + \alpha_3 \left(\frac{1}{\text{THR}}\right) \\ & + \alpha_4 \left(\frac{1}{\text{BFR}}\right) + \alpha_5 (\text{PRT}) + \alpha_6 (\text{PWR}), \end{aligned} \tag{4}$$

where $\alpha_1 \sim \alpha_6$ are normalizing constants.

The Simulated Annealing procedure compares the performance of successive design choices based on the PI. Execution time, processor utilization, processor throughput, and buffer utilization are maximized so that their cost is computed reciprocally; however, power and port traffic are minimized so that the nonreciprocal values are taken. The coefficient values in the cost equation are set according to the desired performance goals of the system.

Optimization of Scheduling Algorithm. The problem of determining optimal scheduling is to determine the optimal start time and execution time of all tasks in the application.

$$\begin{aligned} &\text{minimize}\left(\sum_{i=1}^{N}\left(t_i + e_i\right)\right), \\ &\text{minimize}\left(\sum_{i=1}^{N}\text{COST}\left(\alpha\left(T_i\right)\right)\right), \qquad t_n + e_n \le \text{DL}, \end{aligned} \tag{5}$$

where e_i is the worst-case execution time of each task T_i, DL is the deadline of the application, and t_n and e_n are the start time and worst-case execution time of the last task, respectively, in the application. COST $(\alpha(T_i))$ is the cost function of scheduling the tasks.

Mapping. Mapping is the process of assigning each task in an application to a processor such that the processor executes the task and satisfies the task deadline as well as other constraints (power, throughput, completion time), if any.

For an application $A = \{T_1, T_2 \cdots T_N\}$, where T_i is a task instance in the application, mapping a task, denoted as $M(T_i)$, is defined as an assignment of the processor $\{P_j\} = \{P_1, P_2, \ldots, P_M\} = P$ for the task T_i:

$$M\left(T_i\right) = \left\{P_j\right\}. \tag{6}$$

Application mapping is defined as mapping of all tasks, T_i, in the application A to processors and is denoted as

$$M\left(A\right) = \left\{M\left(T_1\right), M\left(T_2\right) \cdots M\left(T_N\right)\right\}. \tag{7}$$

Optimization of the Mapping Algorithm. The problem in determining optimum mapping is to determine the optimum allocation of all tasks in the application on the processors such that the total execution time of all tasks is minimized. Other factors for optimum mapping include maximizing processor execution time, throughput, minimizing interprocessor communication delay, and balancing workload homogenously on the processors in order to improve utilization

$$\text{minimize}\left(\sum_{i=1}^{N}\left(c_i\right)\right), \qquad c_i = \text{COST}\left(M\left(T_i\right)\right), \tag{8}$$

where c_i is the cost function for optimal mapping of the tasks on the processor.

Power Modeling. In this research, scheduling and mapping algorithms model heterogeneous processor execution and power requirements as follows.

(1) Switch power is consumed whenever a task is routed from input port to output port.

(2) Processor power is modeled with respect to idle-mode (also includes pending, ready, active, and complete) and execution-mode (running) power, both of which are functions of the task execution time, e_i. We also assume that for the idle model, $\Delta\eta = 0$

$$\eta_i\left(T_i\right) = \Delta\eta * e_i, \tag{9}$$

where $\Delta\eta$ is the unit power consumed per unit time of task processing. The total power rating for a processor P_j would be expressed as

$$\eta_j = \Delta\eta * \sum_{i=1}^{n}\left(e_i\right), \tag{10}$$

where n is the number of tasks executed by the processor.

(3) Whenever a task T_k is executed on processor P_l, the execution time e_k and the power consumed by processor η_k are updated as

$$\begin{aligned} \eta_l\left(T_k\right) &= \Delta\eta * e_k, \\ \eta_l &= \Delta\eta * \sum_{i=1}^{n}\left(e_k\right). \end{aligned} \tag{11}$$

(4) Whenever a task is transferred between switches, the communication delay is updated on the Total Task Communication Delay parameter $(\mathcal{T})$ and is expressed as

$$\mathcal{T} = n * \mathfrak{t} * \Delta C, \tag{12}$$

where n is number of switch hopes, task-size $(\mathfrak{t})$ is the size of the task defined as a function of its execution time in time-units, and ΔC is a time constant that is required to transfer single time-unit task for one hop.

(5) For each task T_i, the Task Slack Time S_i and Average Slack Time S are calculated as

$$\begin{aligned} S_i &= d_i - \left[t_i + e_i\right], \\ S &= \frac{\sum_{i=1}^{N}\left(S_i\right)}{N}, \end{aligned} \tag{13}$$

where d_i is the deadline, t_i is the release time, and N is the total number of tasks processed.

Fault-Tolerant System. A fault-tolerant system describes a robust system that is capable of sustaining application processing in the event of component failure. A fault-tolerant algorithm is a process that performs scheduling and mapping on a set of unprocessed tasks for the available processors.

Let an application A with a set of tasks $A = \{T_1, T_2 \cdots T_N\}$ be scheduled, denoted by $\alpha(A)$, and mapped on the set

of processors $P = \{P_1, P_2, \ldots, P_M\}$, denoted as $M(A)$. A fault is defined as a nonrecurring, permanent failure of a processor at time instance t_f during the execution of a task T_f. The application after the occurrence of the fault is expressed as A_f such that $A_f \subset A$, which has a list of tasks $A_f = \{T_g, T_h \cdots T_n\}$ that are not dispatched to processors by the mapper when the processor failure occurs. The set of processors after the failure F is denoted by $P_f = \{P_1, P_2 \cdots P_L\}$ and $P \notin P_f$, where P_f is the failed processor. The proposed fault-tolerant MpSoC framework determines the updated scheduling $\alpha(A_f)$ and updated mapping $M(A_f)$ for a set of unprocessed tasks in the application subset A_f, such that

$$t_n + e_n \leq \mathrm{DL}, \tag{14}$$

where t_n and e_n are the start time and execution time, respectively, of the last scheduled task in A_f.

Optimization of Fault-Tolerant Strategy. Due to the dynamic mapping methodology adopted in the framework, only the tasks dispatched to the failed processor—which includes the task that was executing on the failed processor and the tasks dispatched by the mapper to the failed processor—will need to be rescheduled along with the tasks that were not dispatched by the mapper.

Let us consider an application A_f with a list of n tasks $A_f = \{T_g, T_h \cdots T_n\}$ that have not been dispatched to processors at the occurrence of the fault. Also, let the tasks $T_b = \{T_r \cdots T_t\}$ be the list of tasks in the buffer of the failed processor P_f and let T_p be the task that was being executed during failure by the failed processor. The task list T_{FT} of the fault: tolerant algorithm at time t_f is represented as

$$T_{\mathrm{FT}} = A_f + T_b + T_f. \tag{15}$$

It is also assumed that all the tasks that are being executed by the nonfailed processor complete the execution of the tasks even after the time of failure t_f. Thus, the FT algorithm performs the updated scheduling $\breve{\alpha}(T_{\mathrm{FT}})$ and updated mapping $\dot{M}(T_{\mathrm{FT}})$ on the task set T_{FT}, which satisfies the deadline constraint, DL, of the application A.

4. Framework Modeling and Implementation

The proposed framework adopts a "bottom-up" modular design methodology that gives the benefit of component reusability. Behavioral component models were designed and stored in hardware and software libraries. The framework was implemented in SystemC 2.2/C++ and compiled with g++-4.3 under Linux. The framework is divided into three major components: input or user interface module, core MpSoC simulation module, and the output or performance evaluation module. The hardware modules, such as the processor, switch, dispatcher unit, and multiplexer, were implemented in SystemC; the software modules, such as input parser, scheduler, and task interface classes, were implemented in C++.

Input Module. Such user's data as the number of processors, topology, task graph, processor types, buffer size, switching technology, fault-tolerant mode, and scheduling/mapping specifications are read either by using a command line interface or else from a GUI interface.

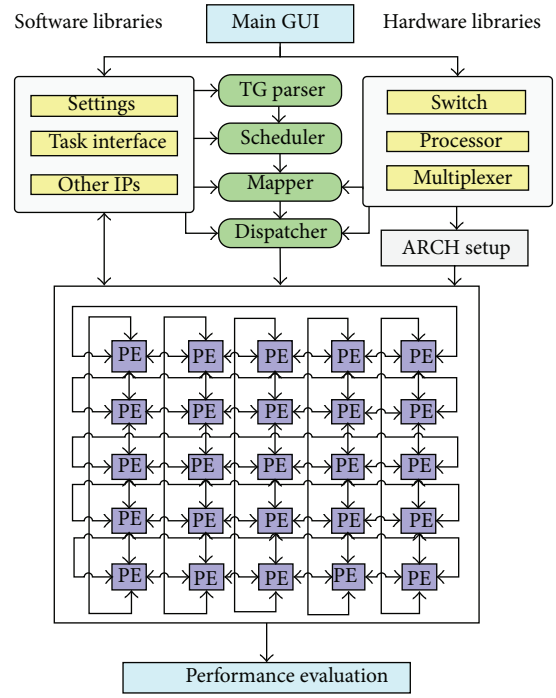

Figure 2: Framework modeling.

Core Simulation Module. The core simulation module is depicted in Figure 2, which shows the software/hardware libraries and control flow between the modules. The simulation module consists of two components: the software and hardware libraries. Software libraries include software components of the framework, for instance, the input parser, scheduler, data store, and task interface. Hardware libraries include hardware components implemented in SystemC, such as the processor, the mapper, the dispatcher, and multiplexers. The processors are implemented as behavioural processors capable of computing the various tasks in the task graphs. The communication cost for a task is dependent on the size, and thus the execution duration, of the task (Section 3). Thus, communication latency is modelled as a function of task size. Heterogeneity is modelled by varying the task processing cost (processing time/task and power/task). All processors are capable of executing all possible operations but with different costs. Processor failure is permanent once it occurs, and only single processor failure is considered. Due to the modular implementation of the simulation module, architecture, topology, routing, scheduling, and mapping methods can be easily changed in MpSoCs to conduct extensive evaluation of design space exploration.

Output Module. The simulation terminates by evaluating the system performance module, which retrieves the various

```
for each task in ATG do{
//set the longest-path-maximum-delay deadline
setDeadline();
//initial solution
task.random_mapToPE(); }
while simulation temp > cooling_threshold
do{
for temperature length{
for task_sub_iteration_length{
while each task is assigned a release time{
//generates the RTW w/ Comm. cost
genRTL();
//rand task from RTL
t_sch = sel_rand(RTL);
//generate release time window
t_sch.genRTW();
//set rel time
t_sch.rel_time = sel_rand(RTW); }
//init and run systemC simulation
sc_start(time);
//calculating objective function value
calcObjFunc();
//accepts or rejects a solution
probFunc();}
for each task in ATG do{
//calculate workload of processors
task.calc_HWDcost();
//new mapping solution
task.HWD_mapToPE(); }
}
//calculate new simulation temperature
calc_simulation_temp();
}
```

Algorithm 1: Pseudocode for the performance driven (PD) scheduling and HWD mapping algorithms.

performance variables from the individual processors, such as the total execution time, power, throughput, port traffic, number of tasks processed, and buffer usage. These values are displayed on the output GUI and stored in the result log file.

4.1. Performance Driven Scheduling Algorithm. This work also proposes an efficient, offline, and performance driven scheduling algorithm based on the Simulate Annealing technique. The performance index (PI), determined by a cost function, is used to determine the optimal schedule. The performance index is a cumulative factor of (1) processor execution time, (2) processor utilization, (3) processor throughput, (4) processor power, (5) processor port traffic, and (6) processor buffer utilization. The problem of determining the optimal schedule is defined as determining the schedule with maximal performance index. The Simulated Annealing algorithm performs (1) random scheduling and mapping of tasks, (2) simulation, and (3) capturing, averaging, and normalizing performance variables; finally, it calculates the performance index. These steps are repeated several times in an effort to increase the PI. During the iterations, if a better PI is found, the state is saved, and the simulation repeats until the temperature reaches the threshold value.

For the problem being considered, the normalizing coefficients used to calculate the PI were set to $\alpha_1 = 300$, $\alpha_2 = 500$, $\alpha_3 = 250$, $\alpha_4 = 125$, $\alpha_5 = 120$, and $\alpha_6 = 50000$. These coefficients were determined after conducting several runs on all benchmarks. Execution time, utilization, throughput, and power factors are key performance indices. Therefore, they are given significant weights (a higher value of coefficient) during normalization. Port traffic and power indices are minimized; consequently, their respective cost function is computed reciprocally. The pseudocode for the performance driven scheduling (PDS) and HWD mapping algorithms is given in Algorithm 1.

4.2. Multiprocessor Task Mapping Strategy. Multiprocessor environments grant the freedom for dynamic task mapping. Thus, the multiprocessor mapping algorithm needs to specify the location where a particular task should be processed. Task mapping in heterogeneous systems is accomplished in two steps: (1) binding and (2) placement. First, the task binding

procedure attempts to find which particular assignment of tasks would lead to effective lower power and faster execution time. During task placement, the mapper considers allocating any two tasks that exchange data more frequently as close to each other as possible. Placement of communicating tasks to adjacent or nearby processors reduces the communication latency, switch traffic, and power dissipation.

Static mapping assigns a particular task to a predetermined processing element offline regardless of the dynamic state of the system components. This, however, does not satisfy the MpSoCs requirement because processing tasks on a heterogeneous architecture introduces workload irregularity on system resources (PEs). Irregular workload distribution is a design issue that degrades the utilization of system. To overcome this problem, tasks should be mapped dynamically, based on the dynamic workload on the processors. The number of tasks in the processors' local memory measures the processor workload.

The dynamic mapper in the proposed framework performs the task allocation as per the configured policy. In order to take heterogeneity of processors into consideration, additional power cost and execution latency cost have been introduced. This emphasizes the fact that processing a particular task on different types of processors incurs different cost (power consumption and execution time). Efficient and optimized heterogeneous mapping algorithms are tailored to minimize this additional cost.

In the framework, various task allocation heuristics were implemented. Given a scheduled task, the task mapping follows one of the below policies.

(1) *Next Available Mapping (NXT AVLB)*. In the Next Available Mapping policy, a scheduled task T_i is mapped on processor P_i, followed by the mapping of the next scheduled task T_{i+1} on processor P_j, provided P_i and P_j are neighbors to each other, as defined by the topology. This policy assigns tasks on successive processor sequentially but does not consider the heterogeneity of the tasks nor the processor workload.

(2) *Homogenous Workload Distribution Mapping (HWD)*. HWD considers the workload on each processor and maps a scheduled task on a processor that has the minimum workload.

(3) Random mapping (RANDOM): rRandom task allocation is used by the PD algorithm to distribute the workload randomly throughout the available processors.

HWD Mapping Approach. Minimizing the idle time of each processor, or maximizing processing time, improves processor utilization. To reduce idle time, tasks should be distributed evenly throughout the available processors. The proposed Homogenous Workload Distribution (HWD) algorithm is tailored to distribute tasks evenly to the available processors in an effort to balance the dynamic workload throughout the processors. The algorithm is a two-step process: (1) probing individual processor buffers through dedicated snooping channels that connects each processor to the mapper and (2) mapping each scheduled task on a processor that has the lowest workload. Workload is a measure of the number of tasks and its respective execution time in the processor buffer. Thus, a processor having the least number of tasks and respective execution time in its buffer has the lowest workload; therefore, the mapper will assign the next scheduled task to this processor.

HWD Mapping Algorithm. Let $\{P_1, P_2 \cdots P_M\}$ be a set of processors, let $\{L_1, L_2 \cdots L_M\}$ be the dynamic task workloads defined by number of tasks on respective processor buffers, and let N be the number of processors. Let $\{T_1, T_2 \cdots T_N\}$ be a set of M tasks and the symbol " $\rightarrow$ " denotes "mapping." Then,

$$\begin{aligned} &\text{A task } T_k \left\langle T_{\text{release-time}}, T_{\text{exec-time}}, T_{\text{deadline}} \right\rangle \longrightarrow P_i \\ &\qquad \text{iff } L_i < L_j \quad \forall \left[L_i, L_j\right] \in \{L_1, L_2 \cdots L_N\}, \end{aligned} \tag{16}$$

where $P_i \in \{P_1, P_2 \cdots P_M\}$ and $T_k \in \{T_1, T_2 \cdots T_N\}$ such that $(e_k + t_k \leq d_k)$.

This approach has three advantages: (1) it minimizes the chance for buffer overflow, (2) the task waiting time in processor buffer is reduced, and (3) utilization is increased because idle processors will be assigned scheduled tasks.

4.3. Fault-Tolerant Methodology. System designs often have flaws that cannot be identified at design time. It is almost impossible to anticipate all faulty scenarios beforehand because processor failure may occur due to unexpected runtime errors. Real-time systems are critically affected if no means of fault tolerance scheme is implemented. Failure to employ fault detection and recovery strategies is often unaffordable in critical mission systems because the error may lead to possible data discrepancies and deadline violation.

Processor failure introduces inevitable overall performance degradation due to (1) the reduced computing power of the system and (2) the overhead involved in applying the fault recovery schemes. Recovery procedures involve tracking the task execution history and reconfiguring the system by using the available processors. This enables application processing to resume when the execution was terminated. During processor failure, application processing is temporarily suspended, and error recovery procedures are applied before application processing is resumed.

Fault-Tolerant Model. Fault modeling can have several dimensions that define the fault occurring time, the fault duration, location of failure, and so forth. The proposed framework adopts the following fault modeling parameters.

(1) It is assumed that the processor failure is permanent. Thus, fault duration time t_{FD} is infinite ($t_{\text{FD}} = \infty$). During failure, the address of the failed processor is completely removed from the mapper address table.

(2) The location specifies where the failure occurred. For tile and torus topology, location is represented in two-dimensional coordinate addresses.

(3) During task processing, any processors may fail at any time instance. Consequently, time of failure is modeled as a random time instance.

In the event of failure, the following procedures are executed.

(1) The mapper removes the failed processor ID from the address table.

(2) Tasks that are scheduled or mapped but not dispatched have to be rescheduled and remapped. The respective new task release time and task address both have to be done again. This is because the execution instance of a task depends on the finish time of its predecessor task, which may possibly have been suspended in the failed processor.

(3) Tasks that are already dispatched to the failed processor can be either (1) in the processor's buffer or (2) in the middle of execution when the failure occurred. These tasks have to be migrated back to the mapper, and their respective rescheduling and remapping have to be done. The scheduler can perform task scheduling concurrently when the nonfaulty processors are executing their respective tasks.

(4) Tasks that are dispatched to other nonfailed processors are not affected.

Due to the dynamic property of the mapper and the fault-tolerant implementation, the cost incurred during the event of processor failure is minimal; overhead incurred is only 2.5% of the total execution time of the task. The only penalty incurred during processor failure is the cost due to task migration and the respective rerescheduling and remapping procedures.

5. Experimental Setup

The heterogeneous MpSoC framework developed in this work was developed using C++ and SystemC programming language. The architectural components such as processing elements (PEs), memory elements, routers and network interfaces are developed using SystemC language and provide cycle accurate simulations. The processing elements are implemented as a simple 32-bit processor that performs simple logic and arithmetic operations. The heterogeneous PE structure in our MpSoC is implemented by altering the type of operation and the execution times of the tasks each PE can execute. The routers implementations consist of a switching matrix with input port buffers. The tasks and data within PEs are transmitted using a data-serializing module of the PE. A typical data transmitted within PEs consists of task commands, task data (if any), and the destination PE IDs (addresses). The data is propagated through the MpSoC system using the *XY* routing algorithm.

The scheduling and mapping algorithm are implemented using C++ language and are interfaced with the SystemC architecture implementations. The application in this work is considered as a task graph. Each task graph consists of processor operations, their execution time, communication cost, and deadline. The application as a whole also has a deadline. The scheduling algorithm parses the application (task graph) and assigns start times for the tasks on the processors based on the set of optimization goals described in the proposed performance driven scheduling algorithm. The scheduled tasks are mapped to their respective processors in the MpSoC system based on the proposed homogenous workload-mapping algorithm. The tasks' information comprising of the task operations, task data, destination PEs (path/address of successor PEs) are assembled and forwarded to the initial PE by the dispatcher module. The performance evaluation module or the monitor module records cycle accurate timing signals in the architecture during tasks execution and communication.

During the simulation of fault-tolerant scenario, the monitor module triggers a PE failure event at a predefined time instance. The dispatcher is informed of the PE failure and based on the mapper table, and the current processing task and the tasks in the buffer of the failed PE are determined. An updated task graph is constructed, and scheduling and mapping on the remaining unexecuted tasks are performed. Tasks currently executing on the PEs are undisturbed. The reassigned tasks are inserted in the PE buffers to satisfy the deadlines.

The implemented MpSoC system has been evaluated extensively by modifying the architecture and topology. The system was evaluated using a set of synthetic task graphs with 50–3000 tasks. The architectural modifications for evaluating the MpSoC include change in type of PEs in the system, mesh and torus topology variations, and the number of PEs in the system (9, 16, 32). The MpSoC system has been used to evaluate the proposed scheduling, mapping, and fault-tolerant algorithms and compared with respective classical algorithms. The performance of the proposed algorithms along with variation in architecture and topology have been evaluated and presented. The performance metrics considered in this work include processor throughput, processor utilization, processor execution time/task and power/task, buffers utilization, and port utilization. Detailed summary of the results and its significance are shown in Section 6.

6. Simulation Results and Analysis

For evaluation of the proposed framework to explore design features variations in scheduling, mapping and fault tolerant algorithms have been considered. The application model adapted is similar to the standard task graph (STG) [50, 51] and E3S benchmark suite [54], which contains random set of benchmarks and application-specific benchmarks. The simulation was conducted with many STG benchmarks having task sizes 50 through 3000. The task graph is characterized by the number of tasks, dependencies of tasks, task size (code size), communication cost (latency for interprocessor transfer), and application deadline. The processor architecture model is derived from the E3S benchmark, where each processor is characterized by the type, tasks it can execute, task execution time, task size (code size), and task power. The processor also has general characteristics like type, overhead cost (preemptive cost), task power/time, and clock rate.

The simulation results presented in this paper follow the following legend: $XY \pm Z$, where X denotes R or T for

Table 1: PE comparison.

No. of PEs	AVG. ETIME/ PE/TASK (μsec)	AVG. UTL/PE	AVG. PWR/PE (μwatts)	AVG. PDP/PE
PE no. 9	4.98	0.48	181.25	907.48
PE no. 16	2.81	0.40	98.34	276.98
PE no. 25	1.89	0.35	63.59	120.06

Table 2: Comparison of task execution times, processor utilization, and throughput of EDF-SA and PDS scheduling algorithms.

	ETIME (μs)	UTL	THR (B/ns)	Std. SA Cost
RAND-50				
EDF-SA	204.26	0.46	0.03	0.16
PDS	**220.03**	**0.55**	**0.03**	**0.19**
SPARSE-96				
EDF-SA	616.51	0.44	0.02	0.15
PDS	**637.89**	**0.49**	**0.02**	**0.17**
RAND-100				
EDF-SA	418.67	0.47	0.03	0.17
PDS	**432.17**	**0.60**	**0.03**	**0.21**
RAND-300				
EDF-SA	1299.45	0.49	0.03	0.17
PDS	**1296.95**	**0.56**	**0.03**	**0.19**
FPPP-334				
EDF-SA	2238.36	0.42	0.02	0.15
PDS	**2337.36**	**0.47**	**0.02**	**0.16**
RAND-500				
EDF-SA	1432.79	0.48	0.02	0.16
PDS	**2125.33**	**0.66**	**0.03**	**0.23**
RAND-1000				
EDF-SA	4163.73	0.52	0.03	0.18
PDS	**4324.58**	**0.58**	**0.03**	**0.20**

torus or tile topology, Y denotes E or S for EDF-SA or PD scheduling algorithm, + or − for presence or absence of fault tolerance algorithm, and Z denotes the number of PEs in the MpSoC system. Performance evaluations on architecture and topological variations were performed but are not included in this paper. Only performance results related to the proposed algorithms are presented.

6.1. Simulation Scenarios. To demonstrate the capabilities of the proposed framework and the algorithms, we performed several simulations on task graph sets and evaluated their effectiveness over the following design abstractions.

(i) Architectural: architecture-based evaluation emphasizes the effect of different PEs in the MpSoC framework. Current implementation of our framework consists of processors of the same architecture that can execute various tasks; that is, each processor has a different set of tasks they can execute. The MpSoC architecture was modified by changing the number of PEs, the types of PEs, and topology. It is important to consider the effect of topology in architecture evaluation because the topology of the MpSoC also dictates the performance of the individual PEs. The effect of such architecture changes is measured in terms of average execution time/PE/task (ETIME) in μsec, average PE utilization (UTL), power/task of PEs (PWR) in μ-Watts, and power delay product (PDP) which are shown in Table 1.

(ii) Scheduling: evaluation of scheduling algorithms over various MpSoC architectures was conducted. Performance of earliest deadline first based simulated annealing (EDF-SA) and the new performance driven (PD) algorithms are compared (Table 2). Performance metrics for evaluating scheduling algorithms are the same as that of architecture evaluations. Also the standardized SA costs are shown.

(iii) Mapping: evaluation of mapping algorithms over various MpSoC architectures was conducted using performance driven scheduling policy. Performance of the proposed HWD, Next Available (NA), and random mapping algorithms is compared as shown in Table 3.

(iv) Fault Tolerant-evaluation and comparison of fault-tolerant (FT) and nonfault-tolerant (NFT) implementation over various MpSoC architectures were conducted, and results for two test cases are shown in Tables 4 and 5.

Table 3: Comparison of HWD and next available and random mapping algorithms (simulation: TS-9).

Mapping	Random	HWD	NXT AVLBL
AVG. ETIME	1417.46	1394.67	1382.5
AVG. UTL	0.48	0.59	0.59
AVG. THR	0.02	0.02	0.02
AVG. PRT	108	107	108
AVG. PWR	59950.3	57996.9	57005
AVG. BFR	5	4	6

6.2. Simulation Results and Explanation. Table 1 shows the average performance metrics obtained for various architectural variations possible in the developed MpSoC framework. The architectural variations include the type of topology (tile or torus), scheduling algorithm (EDF-SA or PDS), mapping algorithm (HWD), fault-tolerant scheme (NFT or FT), and the number of PEs. Table 2 specifically compares the performance of the EDF-SA and PDS scheduling algorithm. Comparison results on total execution time (ECT) in μsec, average processor utilization, and average throughput in bytes/μsec are presented. Other metrics that have less significance have been omitted. Also, comparisons between EDF-SA and PDS algorithms for processor utilization, throughput, and buffer usage evaluations across various architectural

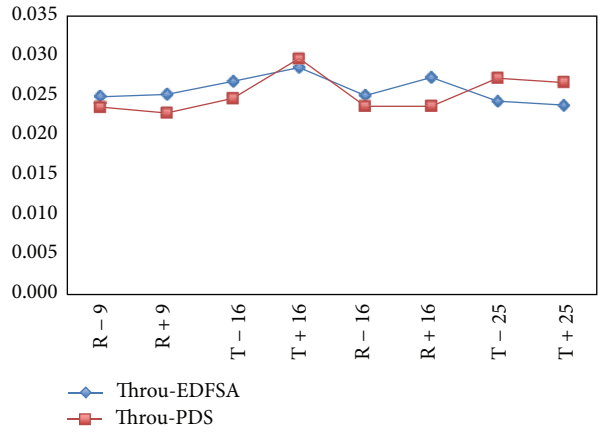

Figure 3: Comparison of processor throughput for EDF-SA and PDS algorithms.

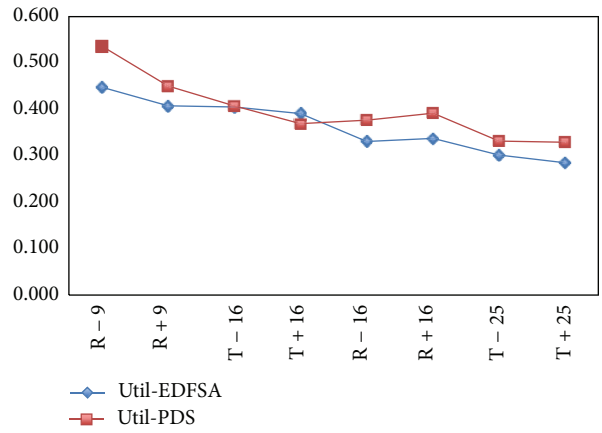

Figure 5: Comparison of processor utilization of EDF-SA and PDS algorithms.

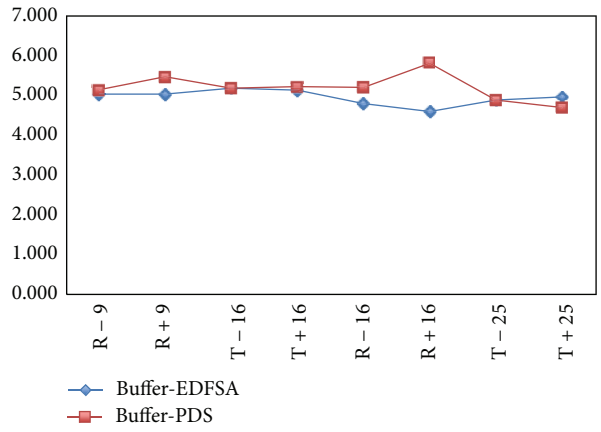

Figure 4: Comparison of buffer usage for EDF-SA and PDS algorithms.

and topological scenarios are shown in Figures 3, 4, and 5. Based on the above table and figures, it can be seen that the utilization, throughput, and buffer usage have increased for a marginal increase in execution time, thereby increasing the overall performance goal or cost of the MpSoC system with PDS algorithm. Table 3 shows the evaluation of the various mapping algorithms. From the table it is clear that the HWD minimizes the use of buffer used during the task execution. The units specified in the tables are as follows: execution time (ETIME) in μsec, processor utilization (UTL) as ratio, throughput (THR) in bytes/nsec, port traffic (PTR) in bytes/nsec, power (PWR) in μWatts, and buffer usage (BFR) in number of 32-bit registers.

The proposed fault-tolerant scheme has been evaluated for a few benchmarks, and two test cases have been presented in Tables 4(a), 4(b), 5(a), and 5(b). For test case 1 (Table 4), a PE fault on PE ID 9 at 100 ns is simulated. Table 4(a) presents the general performance characteristics comparing simulation scenarios without and with PE fault. Table 4(b), presents the detailed performance comparison of significantly affected PEs for without and with fault-tolerant scheme. Also the difference and percentage change (Δ/%) between the simulation scenarios are shown in Table 4(b). Similarly, Tables 5(a) and 5(b) present the second test case for a task graph with 500 tasks operating on tile topology with 81 PEs. The PE fault is introduced on PE ID 80 at 146 ns. Only variations of performance metrics on significantly affected PEs for nonfault and fault simulation are shown. The difference in performance metrics (executions time, port traffic, and power) incurred with the fault-tolerant algorithm/scheme is less significant compared to the original nonfault simulation performance metrics. For example, in Table 5(b), we can see that execution time and processor utilization due to the occurrence of the fault reduce from 397 μs to 91 μs and 0.36 to 0.08, respectively. The tasks assigned to the faulty processor (PE ID = 9) are reassigned to an other processor of the same type using the fault-tolerant algorithm discussed. The increase in execution times and utilization are shown in the table. A marginal increase in cost on other processors has also been observed.

7. Conclusions

The problem designing efficient MpSoC systems require the close scrutiny of the various design choices in order to optimize the performance characteristics. Towards optimizing the scheduling, mapping, and fault-tolerant strategies, a fault-tolerant heterogeneous MpSoCs simulation framework was developed in SystemC/C++ in order to provide a comprehensive simulation tool for designing and verifying proposed methodologies. Three algorithms were proposed: a performance driven (PD) scheduling algorithm, based on Simulated Annealing technique; a strategic Homogenous Workload Distribution (HWD) Mapping algorithm, which

Table 4: Test case no. 1 for fault-tolerant simulation.

(a)

Benchmark	No. of tasks	No. of PEs	PE types
RAND 0000	500	81	8
Schedule	Mapping	Topology	Fault mode
EDF-SA	HWD	TILE	ON
Fault		PE_ID 80 @ 146 μsec	
	W/O fault	W/fault	DIFF
AVG. ETIME	635.59	638.26	−2.67
AVG. UTL	0.25	0.25	0.00
AVG. THR	0.02	0.02	0.00
AVG. PRT	154.00	154.00	0.00
AVG. PWR	26067.00	26129.00	−62.00
AVG. BFR	4.79	4.90	−0.11
MAX EXCT	865.40	875.68	−10.28
TOTAL EXCT	51483.00	51698.96	−215.96
TOTAL PWR	2111417.90	2116413.10	−4995.20
TOTAL BFR	388.00	397.00	−9.00

(b)

PE_ID	ETIME	UTL	THR	PRT	PWR	BFR
1	597.8/646.64 (Δ48.84/8%)	0.24/0.26 (Δ0.02/8%)	0.02/0.02 (Δ0/0%)	1246/1300 (Δ54/4%)	28784.9/30743.7 (Δ1958.8/7%)	9/5 (Δ−4/−44%)
2	693.2/747.74 (Δ54.54/8%)	0.28/0.3 (Δ0.02/8%)	0.02/0.02 (Δ0/0%)	1256/1290 (Δ34/3%)	33939.1/36104.1 (Δ2165/6%)	5/6 (Δ1/20%)
13	538.6/603.29 (Δ64.69/12%)	0.21/0.24 (Δ0.03/12%)	0.02/0.02 (Δ0/−4%)	1208/1282 (Δ74/6%)	22701.9/25943.9 (Δ3242/14%)	8/4 (Δ−4/−50%)
30	434.6/478.51 (Δ43.91/10%)	0.17/0.19 (Δ0.02/10%)	0.03/0.03 (Δ0/−2%)	1201/1275 (Δ74/6%)	15563.3/16941.6 (Δ1378.3/9%)	8/4 (Δ−4/−50%)
36	536/603.82 (Δ67.82/13%)	0.21/0.24 (Δ0.03/12%)	0.02/0.02 (Δ0/−4%)	1112/1211 (Δ99/9%)	20043.8/23566.7 (Δ3522.9/18%)	4/4 (Δ0/0%)
50	689.8/801.7 (Δ111.9/16%)	0.27/0.32 (Δ0.04/16%)	0.02/0.02 (Δ0/0%)	1048/1101 (Δ53/5%)	33068.5/36747.9 (Δ3679.4/11%)	7/6 (Δ−1/−14%)
56	634.2/705.19 (Δ70.99/11%)	0.25/0.28 (Δ0.03/11%)	0.02/0.02 (Δ0/−3%)	1031/1050 (Δ19/2%)	22330.9/25882.6 (Δ3551.7/16%)	7/4 (Δ−3/−43%)
64	735.8/804.51 (Δ68.71/9%)	0.29/0.32 (Δ0.03/9%)	0.02/0.02 (Δ0/−1%)	999/1116 (Δ117/12%)	31208.9/33490.9 (Δ2282/7%)	3/8 (Δ5/167%)
72	807.2/875.68 (Δ68.48/8%)	0.32/0.35 (Δ0.03/8%)	0.01/0.01 (Δ0/0%)	881/965 (Δ84/10%)	34891.6/37150.1 (Δ2258.5/6%)	7/4 (Δ−3/−43%)
80*	**772/149.2 (Δ−622.8/−81%)**	**0.31/0.06 (Δ−0.25/−81%)**	**0.02/0.01 (Δ0/−14%)**	**883/290 (Δ−593/−67%)**	**30763.9/6858 (Δ−23905.9/−78%)**	**3/2 (Δ−1/−33%)**

considers dynamic processor workload; and a fault-tolerant (FT) methodology to deliver a robust application processing system.

Extensive simulation and evaluation of the framework as well as of the proposed and classical algorithms were conducted using task graph benchmarks. Simulation results on all performance indices for different scenarios were evaluated and discussed. For the scheduling space, the PD heuristic has shown better overall performance than EDF, specifically for small number of processors. Fault-tolerant

Table 5: Test case no. 2 for fault-tolerant simulation.

(a)

Benchmark	No. of tasks	No. of PEs	PE types
RAND 0000	100	36	6
Scheduding	Mapping	Topology	Fault mode
EDF-SA	HWD	TILE	ON
Fault		PE_ID 9 @ 100 μsec	
	W/O fault	W/fault	DIFF
AVG. ETIME	423.16	427.37	−4.21
AVG. UTL	0.38	0.39	0.00
AVG. THR	0.02	0.02	0.00
AVG. PRT	41.00	41.00	0.00
AVG. PWR	18553.70	18736.40	−182.70
AVG. BFR	5.31	4.94	0.36
MAX EXCT	520.60	533.70	−13.10
TOTAL EXCT	15233.60	15385.16	−151.56
TOTAL PWR	667932.79	674509.41	−6576.62
TOTAL BFR	191.00	178.00	13.00

(b)

PE_ID	ETIME	UTL	THR	PRT	PWR	BFR
3	379.8/425.4 (Δ45.6/12%)	0.34/0.38 (Δ0.04/12%)	0.02/0.02 (Δ0/−1%)	353/367 (Δ14/4%)	17365.2/19309.1 (Δ1943.9/11%)	3/6 (Δ3/100%)
5	441.2/509.74 (Δ68.54/16%)	0.4/0.46 (Δ0.06/15%)	0.02/0.02 (Δ0/−4%)	317/324 (Δ7/2%)	20514.3/24301.6 (Δ3787.3/18%)	6/4 (Δ−2/−33%)
***9**	**397.4/91 (Δ−306.4/−77%)**	**0.36/0.08 (Δ−0.28/−77%)**	**0.02/0.02 (Δ0/−3%)**	**323/180 (Δ−143/−44%)**	**15788.7/2993.26 (Δ−12795.44/−81%)**	**6/8 (Δ2/33%)**
19	356.8/407.77 (Δ50.97/14%)	0.32/0.37 (Δ0.04/14%)	0.02/0.02 (Δ0/−2%)	278/320 (Δ42/15%)	15606.8/17142.9 (Δ1536.1/10%)	4/3 (Δ−1/−25%)
22	266.2/329.47 (Δ63.27/24%)	0.24/0.3 (Δ0.06/23%)	0.03/0.03 (Δ0/−9%)	269/283 (Δ14/5%)	8951.62/11269.9 (Δ2318.28/26%)	9/3 (Δ−6/−67%)
23	354.6/455.78 (Δ101.18/29%)	0.32/0.41 (Δ0.09/28%)	0.02/0.02 (Δ0/−3%)	270/300 (Δ30/11%)	12774/17433.6 (Δ4659.6/36%)	6/4 (Δ−2/−33%)
29	462/533.7 (Δ71.7/16%)	0.42/0.48 (Δ0.06/15%)	0.02/0.02 (Δ0/−3%)	224/256 (Δ32/14%)	20069.4/23731.2 (Δ3661.8/18%)	2/7 (Δ5/250%)

evaluations showed that throughput, buffers utilization, execution time/task, and power/task factors are not significantly affected even after processor failure occurs. A fault-tolerant scheme showed a small decrease in processor utilization. Tile topology showed better utilization and throughput; however, torus topology showed significantly better performance with respect to execution time/task and power/task. A number of processor comparisons showed a proportional decrease in utilization, execution time, and power when the number of processors was increased. However, throughput and buffer utilization remained almost identical. Executing highly heterogeneous tasks resulted in higher power and latency costs. Finally, the proposed HWD algorithm evenly distributed the execution workload among the processors, which improves the processor overall performance, specifically processor and buffer utilization.

The proposed algorithms on the MpSoC framework are currently evaluated only with synthetic task graphs. We would like to evaluate the algorithms for application benchmarks. The PE implemented is a basic 32-bit behavioral processor. We would like to implement industry standard processor like ARM or OpenRISC processors as our PEs. Also, during scheduling communication, cost values are derived from the execution time of the tasks. Future work that

considers actual communication cost needs to be explored. Work related to router and buffer failures will also be considered.

References

[1] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping applications to tiled multiprocessor embedded systems," in *Proceedings of the 7th International Conference on Application of Concurrency to System Design (ACSD '07)*, pp. 29–40, July 2007.

[2] C. Neeb and N. Wehn, "Designing efficient irregular networks for heterogeneous systems-on-chip," *Journal of Systems Architecture*, vol. 54, no. 3-4, pp. 384–396, 2008.

[3] S. Manolache, P. Eles, and Z. Peng, *Real-Time Applications with Stochastic Task Execution Times, Analysis and Optimization*, Springer, 1st edition, 2007.

[4] W. Wolf, *High-Performance Embedded Computing*, Elsevier, 20072007.

[5] A. A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips*, Morgan Kaufmann Series in Systems on Silicon, 2005.

[6] J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, *Interconnect-Centric Design for Advanced SoC and NoC*, Kluwer Academic, 2004.

[7] N. D. Thai, "Real-time scheduling in distributed systems," in *Proceedings of the IEEE International Conference on Parallel Computing in Electrical Engineering (PARELEC '02)*, pp. 165–170, 2002.

[8] R. Marculescu, U. Y. Ogras, L. S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3–21, 2009.

[9] G. De Micheli and L. Benini, *Networks on Chips: Technology and Tools Systems on Silicon*, Academic Press, 2006.

[10] R. Marculescu and P. Bogdan, "The Chip is the network: toward a science of network-on-chip design," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 4, pp. 371–461, 2007.

[11] S. Stuijk, T. Basten, M. Geilen, A. H. Ghamarian, and B. Theelen, "Resource-efficient routing and scheduling of time-constrained network-on-chip communication," in *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD '06)*, pp. 45–52, September 2006.

[12] J. Hu and R. Marculescu, "Communication and task scheduling of application-specific networks-on-chip," *Proceedings of the IEEE*, vol. 152, no. 5, pp. 643–651, 2005.

[13] G. Varatkar and R. Marculescu, "Communication-aware task scheduling and voltage selection for total systems energy minimization," in *International Conference on Computer Aided Design (ICCAD '03)*, pp. 510–517, San Jose, Calif, USA, November 2003.

[14] S. Zhang, K. S. Chatha, and G. Konjevod, "Approximation algorithms for power minimization of earliest deadline first and rate monotonic schedules," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '07)*, pp. 225–230, August 2007.

[15] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *Proceedings of the International Symposium on Low Electronics and Design (ISLPED '01)*, pp. 46–51, August 2001.

[16] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Iterative schedule optimization for voltage scalable distributed embedded systems," *Journal ACM Transactions on Embedded Computing Systems*, vol. 3, no. 1, pp. 182–217, 2004.

[17] D. Shin and J. Kim, "Power-aware communication optimization for networks-on-chips with voltage scalable links," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '04)*, pp. 170–175, September 2004.

[18] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," in *Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP '07)*, pp. 34–40, May 2007.

[19] H. J. Ahn, M. H. Cho, M. J. Jung, Y. H. Kim, J. M. Kim, and C. H. Lee, "UbiFOS; a small real-time operating system for embedded systems," *ETRI Journal*, vol. 29, no. 3, pp. 259–269, 2007.

[20] C. Ostler and K. S. Chatha, "An ILP formulation for system-level application mapping on network processor architectures," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '07)*, pp. 99–104, April 2007.

[21] H. M. Harmanani and R. Farah, "A method for efficient mapping and reliable routing for NoC architectures with minimum bandwidth and area," in *Proceedings of the Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference (NEWCAS-TAISA '08)*, pp. 29–32, June 2008.

[22] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, 2005.

[23] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, pp. 896–901, February 2004.

[24] P. Poplavko, T. Basten, M. Bekooij, J. Van Meerbergen, and B. Mesman, "Task-level timing models for guaranteed performance in multiprocessor networks-on-chip," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '03)*, pp. 63–72, November 2003.

[25] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based NoC architectures," in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '04)*, pp. 182–187, September 2004.

[26] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 387–392, August 2005.

[27] P. Brucker, *Scheduling Algorithms*, 5th edition, 2006.

[28] X. Zhu and W. Qin, "Prototyping a fault-tolerant multiprocessor SoC with run-time fault recovery," in *Proceedings of the 43rd Annual Design Automation Conference (DAC '06)*, pp. 53–56, 2006.

[29] R. Holsmark, M. Palesi, and S. Kumar, "Deadlock free routing algorithms for mesh topology NoC systems with regions," in *Proceedings of the 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD '06)*, pp. 696–703, September 2006.

[30] M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI: Emerging Trends in VLSI Systems Design*, pp. 46–51, February 2004.

[31] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 818–831, 2005.

[32] F. Angiolini, D. Atienza, S. Murali, L. Benins, and G. De Micheli, "Reliability support for on-chip memories using Networks-on-Chip," in *Proceedings of the 24th International Conference on Computer Design (ICCD '06)*, pp. 389–396, October 2006.

[33] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide, "Immunet: a cheap and robust fault-tolerant packet routing mechanism," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, pp. 198–209, June 2004.

[34] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini, "Fault tolerance overhead in network-on-chip flow control schemes," in *proceedings of the 18th Symposium on Integrated Circuits and Systems Design (SBCCI '05)*, pp. 224–229, September 2005.

[35] P. Beerel and M. E. Roncken, "Low power and energy efficient asynchronous design," *Journal of Low Power Electronics*, vol. 3, no. 3, pp. 234–253, 2007.

[36] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston, "A lightweight fault-tolerant mechanism for network-on-chip," in *Proceedings of the 2nd IEEE International Symposium on Networks-on-Chip (NOCS '08)*, pp. 13–22, April 2008.

[37] P. Bogdan, T. Dumitraş, and R. Marculescu, "Stochastic communication: a new paradigm for fault-tolerant Networks-on-chip," *VLSI Design*, vol. 2007, Article ID 95348, 17 pages, 2007.

[38] S. Murali and G. De Micheli, "SUNMAP: a tool for automatic topology selection and generation for NoCs," in *Proceedings of the 41st Design Automation Conference (DAC '04)*, pp. 914–919, June 2004.

[39] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "Xpipescompiler: a tool for instantiating application specific networks on chip," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, pp. 884–889, February 2004.

[40] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, "NNSE: nostrum network-on-chip simulation environment," in *Proceedings of the Swedish System-on-Chip Conference (SSoCC '03)*, April 2005.

[41] S. Mahadevan, M. Storgaard, J. Madsen, and K. Virk, "ARTS: a system-level framework for modeling MPSoC components and analysis of their causality," in *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS '05)*, pp. 480–483, September 2005.

[42] P. G. Paulin, C. Pilkington, and E. Bensoudane, "StepNP: a system-level exploration platform for network processors," *IEEE Design and Test of Computers*, vol. 19, no. 6, pp. 17–26, 2002.

[43] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *Proceedings of the Design, Automation and Test in Europe (DATE '05)*, pp. 246–251, March 2005.

[44] M. Coppola, S. Curaba, M. D. Grammatikakis, G. Maruccia, and F. Papariello, "OCCN: a network-on-chip modeling and simulation framework," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, DATE 04*, vol. 3, pp. 174–179, February 2004.

[45] C. A. Zeferino and A. A. Susin, "SoCIN: a parametric and scalable Network-on-Chip," in *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design*, pp. 169–174, 2003.

[46] H. Orsila, E. Salminen, M. Hännikäinen, and T. D. Hämäläinen, "Evaluation of heterogeneous multiprocessor architectures by energy and performance optimization," in *Proceedings of the International Symposium on System-on-Chip (SOC '08)*, pp. 1–6, November 2008.

[47] H. Orsila, T. Kangas, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen, "Automated memory-aware application distribution for Multi-processor System-on-Chips," *Journal of Systems Architecture*, vol. 53, no. 11, pp. 795–815, 2007.

[48] F. Paterna, L. Benini, A. Acquaviva, F. Papariello, and G. Desoli, "Variability-tolerant workload allocation for MPSoC energy minimization under real-time constraints," in *Proceedings of the IEEE/ACM/IFIP 7th Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '09)*, pp. 134–142, October 2009.

[49] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 363–374, 2008.

[50] Z. Lide, L. S. Bai, R. P. Dick, S. Li, and R. Joseph, "Process variation characterization of chip-level multiprocessors," in *Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC '09)*, pp. 694–697, New York, NY, USA, July 2009.

[51] "Embedded System Synthesis Benchmarks Suite (E3S)," http://ziyang.eecs.umich.edu/~dickrp/e3s/.

[52] C. Ababei and R. Katti, "Achieving network on chip fault tolerance by adaptive remapping," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS '09)*, pp. 23–29, May 2009.

[53] O. Derin, D. Kabakci, and L. Fiorin, "Online task remapping strategies for fault-tolerant network-on-chip multiprocessors," in *Proceedings of the 5th ACM/IEEE International Symposium on Networks-on-Chip (NOCS '11)*, pp. 129–136, May 2011.

[54] "Standard Task Graph (STG)," http://www.kasahara.elec.waseda.ac.jp/schedule/.