Task Mapping on NoC-Based MPSoCs with Faulty Tiles

Evaluating the Energy Consumption and the Application Execution Time

Alexandre M. Amory, César A. M. Marcon,
Fernando G. Moraes
FACIN – Faculdade de Informática –
PUCRS Catholic University
Porto Alegre, Brazil
{alexandre.amory, cesar.marcon,
fernando.moraes}@pucrs.br

Marcelo S. Lubaszewski
PPGC – Instituto de Informática –
UFRGS Federal University
Porto Alegre, Brazil
luba@eletro.ufrgs.br

Abstract— **The use of spare tiles in a networks-on-chip based multi-processor chip can improve the yield, reducing the cost of the chip and maintaining the system functionality even if the chip is defective. However, the impact of this approach on application characteristics, such as energy consumption and execution time, is not documented. For instance, on one hand the application tasks might be mapped onto any tile of a defect-free chip. On the other hand, a chip with a defective tile needs special task mapping that avoid fault tiles. This paper presents a task mapping aware of faulty tiles, where an alternative task mapping can be generated and evaluated in terms of energy consumption and execution time. The results show that faults on tiles have, on average, a small effect on energy consumption but no significant effect on execution time. It demonstrates that spare tiles can improve yield with a small impact on the application requirements.**

Keywords: MPSoC, task mapping, yield, energy consumption, execution time.

I. Introduction

A multiprocessor system-on-chip (MPSoC) is typically a very large scale integrated system that incorporates most or all the components necessary for an application, including multiple processors [1]. A network-on-chip (NoC) is the preferable intrachip communication infrastructure for MPSoCs due to its superior performance, scalability, and modularity. MPSoCs that use NoCs as the communication infrastructure are also called NoC-based MPSoCs.

NoCs can consume more than one third of the total chip energy [2][3]. On the other hand, the shrinking feature-sizes of newer technologies and the supply voltage scaling [4][5] increases the defect rate in the chip manufacturing and reduces the yield. High manufacturability, low latency and energy consumption are conflicting design goals, thus all these requirements have to be *jointly evaluated* to optimize a NoC-based MPSoC design.

The *task mapping* problem determines an association of each application task to a tile to minimize some given cost function. This paper presents a tool that finds an optimal task mapping in terms of energy consumption and application execution time, given a set of tiles with manufacturing defects. This way, even chips with defects can be sold, perhaps with some performance degradation, targeting low-end markets.

The *goals of this paper* are to present the aforementioned task mapping tool and to investigate the energy consumption and application execution time degradations assuming different application classes. The *contributions* of this paper are (*i*) a task mapping tool for NoC-based MPSoC, which consider faulty tiles to perform the mapping; (*ii*) the evaluation of energy consumption and application execution time under the presence of faulty tiles; (*iii*) a statistical method to generate fault scenarios for very large SoCs.

The paper is organized as follows: Section II presents motivation, usage of the proposed approach, and main assumptions. Section III describes the related work. Section IV describes the task mapping tool and its models. Section V describes the experimental setup, the evaluated applications, and the fault scenarios. Section VI discusses the results. Section VII concludes the paper.

II. Preliminaries

A. System Model and Assumptions

This paper assumes that the target MPSoC consists of a set of identical (or homogeneous) tiles connected by a mesh-based NoC with XY routing algorithm. Each tile contains three main components: a network interface, a processor, and a memory block. A tile supports one task only (no multitasking). This system model is equivalent, for instance, to the underlying model of HeMPS MPSoC [6] with the Hermes NoC [7].

The present work assumes faults only on the tiles since we assume that the tile area is at least 90% of the router. Therefore, the communication infrastructure is assumed faulty-free. A faulty tile is completely shutdown, thus it does not consume energy and generate traffic in the network.

The faults are result of defects created during the chip manufacturing. These defects are expected to be more common due to the evolution of deep submicron technologies, thus multiple faults on the chip are considered. The proposed task mapping is executed in design time for several fault scenarios, such that an overall picture of the relationship between the fault location and the performance metrics can be draw.

B. Motivating Example

Redundant hardware is commonly used to tackle the yield problem. It has been successfully applied to all sorts of regular and repetitive hardware, like different types of memories, programmable logic array, field programmable gate array, and recently to MPSoCs [5]. In the context of MPSoCs, the application task located in a faulty tile can be *mapped* (in design time) or *migrated* (in run-time) to a spare tile, keeping the chip functionality.

Shamshiri and Cheng [5] proposed a yield and cost analysis framework employed to evaluate the use of spare tiles in MPSoCs. This one can be used to determine the amount of redundancy required to achieve a minimum cost. For instance, given some input parameters detailed in [5], the yield of a block is 94%, the NoC link is 72%, resulting in a system yield of just 21% for a 3x3 mesh NoC, i.e. there is probability of 79% of having at least one faulty block in the system. By including three spare tiles to

978-1-4577-0660-8/11/$26.00 ©2011 IEEE

the system, increasing the number of tiles from 9 to 12, the system yield increases to 99% since only 9 out of 12 tiles are actually required to have a functional system. Moreover, the manufacturing cost is 3.2 times less than the original system, since the additional silicon area of the spare tiles is compensated by the increased yield.

Given these motivating results, we decided to investigate the use of spare tiles by *evaluating the side effects of multiple faulty tiles on the energy consumption and application execution time.*

C. Usage of the Proposed Approach

Figure 1 illustrates the proposed test approach, which starts as soon as the chip is manufactured. If the tested chip fails, a diagnose step is performed to locate the faulty tiles.

Let *n* be the number of system tiles and *m* be the number of necessary tiles to implement the systems functionality, then *n-m* is the number of spare tiles. If the number of faulty tiles is lower or equal than *n-m*, the place of these faulty tiles is sent to task mapping tool, otherwise the faulty chip is discarded. The task mapping tool, presented in Section IV, loads a NoC model and the application task graph to determine the new task mapping avoiding the faulty tiles. Finally, the tool is able to estimate the energy consumption and the application execution time of the resulting task mapping. Depending on the resulting overhead, chips with up to *n-m* faulty tiles can still be sent to the market, perhaps targeting low-end markets.

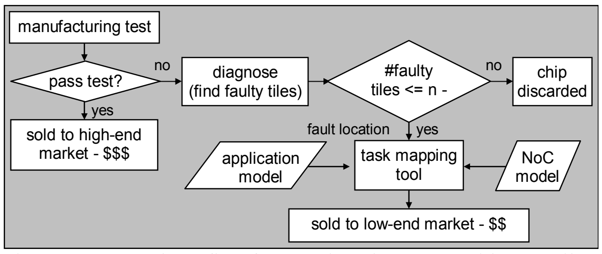


Figure 1. Proposed test flow for NoC-based MPSoCs with spare tiles.

III. Related Work

There are several papers presenting approaches to improve the reliability of NoC-based SoCs. These papers can be broadly classified (these classes are definitely not exhaustive) in: (*i*) fault tolerant circuitry for NoCs and MPSoCs [8][9]; (*ii*) fault tolerant NoC routing algorithm used to explore different routes of packets in case of network faults [10]; (*iii*) system-level reliability assessment [5]; (*iv*) system-level reliability co-optimization [11].

This paper fits best in the system-level reliability co-optimization category, where two main approaches are found: dynamic approaches executed in run-time; or static approaches executed in design time. The *dynamic system-level reliability co-optimization* approach is commonly based on on-line *task mapping* and *task migration* to better accommodate new incoming tasks on the fly assuming the chip might have faults. It can also be used to react, for instance, upon a run-time fault which could have been generated by transient effects or permanent faults due to wear out or aging [15][16]. In this case the tasks located at the faulty resources are moved in runtime to healthy resources. These approaches are out of the scope of this paper since the goal is to improve the yield of the chip manufacturing. Manufacturing defects are not dynamic and they do not appear during run-time.

For this reason this paper is best related to *static system-level reliability co-optimization* approaches, based on static task scheduling executed in design time. Typically these approaches are largely used in design space exploration targeting the optimization of metrics, such as application execution time, latency, thermal constraints, and energy consumption [12][13][14]. Recently these approaches also co-optimize reliability related metrics.

Manolache *et al.* [11] address the reliability problem at application level. They propose a way to combine spatially and temporally redundant message transmission, where energy and latency overhead are minimized.

Tornero *et al.* [17] propose a multi-objective optimization strategy, which minimizes energy consumption and maximizes a robustness index, called path diversity, which explores the multiple paths between a pair of nodes. In case of a faulty link, a NoC with adaptive or source-based routing algorithms could explore these multiples paths, improving the chip robustness.

Choudhur *et al.* [18] introduce a new task mapping, whose objective is to minimize the variance of the system power and latency when faults occur and maximizes the probability that the actual system will work when deployed.

Huang *et al.* [19] argue that some processors might age much faster than others might, reducing the system's lifetime. They proposed an analytical model to estimate the lifetime reliability of MPSoCs. This model is integrated to a task mapping algorithm that minimizes the energy consumption of the system and satisfies system lifetime reliability constraint. Huang and Xu [20] expand their previous task mapping tool [19] to support multi-mode embedded systems. Huang and Xu [21] argue that exponential lifetime distribution can be inaccurate, thus they further refine the lifetime reliability model to support arbitrary lifetime distributions, improving the accuracy of the simulation results.

IV. Task Mapping Aware of Faulty Tiles

The CAFES task mapping framework [22] is composed of high-level models, algorithms and tools, whose goal is to map application tasks onto the target architecture tiles aiming to save energy and to minimize the execution time. Figure 2 illustrates a partial mapping flow and the main elements used here.

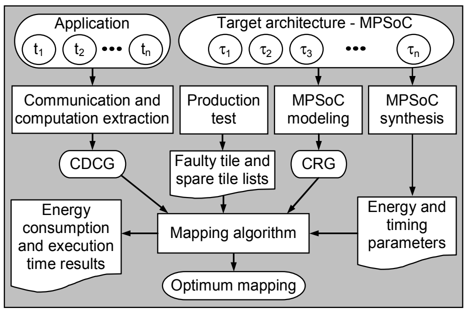

Application
t1 t2 ••• tn
Target architecture - MPSoC
τ1 τ2 τ3 ••• τn
Communication and computation extraction
Production test
MPSoC modeling
MPSoC synthesis
CDCG
Faulty tile and spare tile lists
CRG
Energy consumption and execution time results
Mapping algorithm
Energy and timing parameters
Optimum mapping

Figure 2. Mapping flow used to obtain optima application mappings.

Based on the description of an application already partitioned into tasks t_i, the designer may extract the relevant computation and communication aspects.

Communication Dependence and Computation Graph (CDCG) is a model used to describe the application. Each CDCG vertex models a communication with the source and target task, the communication volume and the computation time - the period between all dependences are solved and communication beginning. CDCG edges represent the communication dependence, i.e. all vertices are connected to each dependence with an edge. The CDCG is similar to a schedule graph, but focusing on communication aspects instead of computation, which enables to explore several requirements of communication architecture easily.

Figure 3 depicts a small example of CDGC, containing three communications $\{C_1, C_2$ and $C_3\}$. C_1 and C_3 are concurrent communications and both do not have dependences, since dependences of the Start vertex (***d_{Start_1}*** and ***d_{Start_3}***) are always solved. Thus, ***C_1*** and ***C_3*** communications start immediately after the respective computation time: 10 and 20 clock cycles, respectively. Communication ***C_1*** states that t_1 send 100 bytes to t_2 and ***C_3*** states that t_3 send 100 bytes to t_1. As soon as the last byte of ***C_1*** is inserted into the NoC, ***d_{1_2}*** is solved. On the other hand, ***d_{3_2}*** is solved only when the last byte of ***C_3*** communication arrives to the processor where t_1 is mapped.

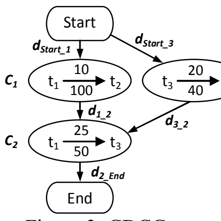


Figure 3. CDCG example.

The target architecture topology is modeled by means of a *Communication Resource Graph* (CRG), which consists of tiles (graph nodes) and links (graph edges). The energy and execution time parameters are extracted from the target architecture synthesized to a given technology. The faulty tile list is generated by the diagnostic flow presented in Figure 1. According to the application description, NoC energy parameters, NoC execution time parameters, NoC topology and the faulty tile list, the task mapping tool estimates the NoC energy consumption and the application execution time of different mappings, enabling to evaluate the impact of faulty tiles. The next sections detail the underlying algorithms and the timing and energy models.

A. Mapping Algorithm

As stated before, the *mapping problem* here consists in finding an association of each application task to a given processor – placed in a given tile – that minimizes the global energy consumption and the application execution time. Let **n** be the number of tiles, this problem allows **n**! possible solutions. Given that future MPSoCs may contain hundreds of tiles, an exhaustive search of the solution space is clearly unfeasible. Thus, optimal implementations of such SoCs require the development of efficient mapping heuristics.

Exhaustive analyses of some small applications mapped on NoC-based MPSoCs show that task mapping is clearly a problem with self-similarity [23] behavior. In other words, *there are several very different mappings with the same cost* – i.e. the same energy consumption and execution time. Therefore, exploring, not all, but very different random mappings followed by some refinements (new mapping with few changes), normally result on an optimized solution. Due to two nested loops – an external one, which looks for very different solutions and an internal one, which looks for a local minimum – Simulated Annealing (SA) is an algorithm very well adequate to find solutions for self-similar problems.

Our SA mapping algorithm searches for mappings that result in an MPSoC with minimum energy consumption and low execution time. To explore these requirements in the same cost function, the execution time requirement is expressed in terms of energy consumption. Therefore, the static power dissipation is multiplied by the application execution time (**texec**) performing the static portion of energy consumption, which is detailed in Section IV.B. As a result, both dynamic and static energy consumption are considered to compute the mapping cost function.

To improve the yield, the SA algorithm searches for mappings with minimum cost avoiding the ones that are marked as spares or faulty. However, when a tile is marked as faulty, the algorithm replaces the faulty tile with a spare tile, which is faulty-free.

B. Timing Model

The *total packet delay* ($\mathbf{d_{ijq}}$) of a wormhole routing algorithm is composed by the *routing delay* ($\mathbf{dR_{ijq}}$) and by the *packet delay* ($\mathbf{dP_{ijq}}$) of the remaining flits. The routing delay is the time necessary to create the communication path, which is determined during the traffic of the packet header. The packet delay depends on the number of remaining flits. Let $\mathbf{n_{abq}}$ be the number of flits of the **q**-th packet from $\mathbf{p_a}$ to $\mathbf{p_b}$, obtained by dividing $\mathbf{w_{abq}}$ by the link width. Let λ be the period of a clock cycle, and let $\mathbf{t_r}$ be the number of cycles needed to route a packet inside a router. In addition, let $\mathbf{t_l}$ be the number of cycles needed to transmit a flit through a link (between tiles or between a processor and a router). The routing delay ($\mathbf{dR_{ijq}}$) and the packet delay ($\mathbf{dP_{ijq}}$) of the **q**-th packet from tile τ_i to tile τ_j, are represented in Equations (1) and (2), considering that a packet goes through η routers without contention. Contentions can only be determined at execution time.

$$\mathbf{dR_{ijq}} = (\eta \times (\mathbf{t_r} + \mathbf{t_l}) + \mathbf{t_l}) \times \lambda \tag{1}$$

$$\mathbf{dP_{ijq}} = (\mathbf{t_l} \times (\mathbf{n_{abq}} - 1)) \times \lambda \tag{2}$$

Equation (3) expresses the total packet delays ($\mathbf{d_{ijq}}$) – packet latency, obtained from the sum of ($\mathbf{dR_{ijq}}$) and ($\mathbf{dP_{ijq}}$).

$$\mathbf{d_{ijq}} = (\eta \times (\mathbf{t_r} + \mathbf{t_l}) + \mathbf{t_l} \times \mathbf{n_{abq}}) \times \lambda \tag{3}$$

For example, when applying Equation (3) in a packet with 10 flits ($\mathbf{n_{abq}} = 10$), which is sent from tile τ_1 to tile τ_2 (two neighbors tiles, i.e. $\eta = 2$), and considering $\lambda = 1$ns, $\mathbf{t_r} = 3$ and $\mathbf{t_l} = 1$ clock cycles, then 18ns is the packet latency.

The application execution time (**texec**) depends on both the application computation and communication. However, a simple equation does not express **texec**, since several communications and computations are many times parallel. In addition, some communications may compete for the same communication resource (e.g. links and buffers) at same time, which may cause contentions increasing the overall execution time. Contentions also make a single equation more complex. Therefore, **texec** is computed during the mapping algorithm execution, which uses several times the $\mathbf{d_{ijq}}$ and time expend in each computation.

C. Energy Model

The dynamic energy consumption is modeled using the concept of *bit energy* (**EBit**), similarly to the model described in [24]. For several communication architectures, **EBit** can be expressed as a function of four variable quantities, as depicted by Equation (4).

$$\mathbf{EBit} = \text{function}(\mathbf{Es}, \mathbf{Eb}, \mathbf{Ec}, \mathbf{El}) \tag{4}$$

Es is the dynamic energy consumption of a single bit on wires and on logic gates of each router. **Eb** is the bit dynamic energy consumption on router buffers. **Ec** is the dynamic energy consumption of a single bit on links between routers and the local module. **El** is the bit dynamic energy consumption on the links between routers.

Equation (5) illustrates how **EBit** models a 2D direct mesh NoC. It computes the dynamic energy consumed by a bit passing in such a NoC from tile **i** (τ_i) to tile **j** (τ_j), where η_{ij} corresponds to the number of routers that the bit traverses.

$$\mathbf{EBit_{ij}} = \eta_{ij} \times (\mathbf{Es}+\mathbf{Eb}) + 2 \times \mathbf{Ec} + (\eta_{ij} - 1) \times \mathbf{El} \qquad (5)$$

Let $\mathbf{w_{abq}}$ be the total amount of bits of a packet $\mathbf{p_{abq}}$ going from $\mathbf{p_a}$ to $\mathbf{p_b}$ (i.e. processors **a** and **b**, correspondingly), which are mapped on tiles τ_i and τ_j, respectively. Then, the dynamic energy consumed by the all **k** packets of $\mathbf{p_a} \rightarrow \mathbf{p_b}$ communications is given by Equation (6).

$$\mathbf{EBit_{ab}} = \sum_{q=1}^{k} w_{abq} \times EBit_{ij} \qquad (6)$$

Hence, Equation (7) gives the total dynamic energy consumed by the NoC (**EDyNoC**) and **y** represents the total number of communication between different processors $\mathbf{p_a}$ to $\mathbf{p_b}$.

$$\mathbf{EDyNoC} = \sum_{i=1}^{y} EBit_{ab}\ (\,)\ \forall\ \mathbf{p_a}, \mathbf{p_b} \in \text{processors set} \qquad (7)$$

The *static power dissipation* of each router (**PRouter**) is proportional to the number of gates that compose the router and it can be estimated by electrical simulation. With **n** representing the number of tiles, Equation (8) computes *NoC static power dissipation* (**PNoC**).

$$\mathbf{PNoC} = \mathbf{n} \times \mathbf{PRouter} \qquad (8)$$

Using **texec** explained in Section IV.B, Equation (9) computes *NoC static energy consumption* (**EsNoC**).

$$\mathbf{EsNoC} = \mathbf{PNoC} \times \mathbf{texec} \qquad (9)$$

Finally, Equation (10) gives the overall energy consumption at the NoC (**ENoC**) that considers the static and dynamic effects, which SA algorithm uses as cost function to search for optima mappings.

$$\mathbf{ENoC} = \mathbf{EsNoC} + \mathbf{EDyNoC} \qquad (10)$$

D. Model Calibration

The Hermes NoC [7], configured with 16-bit phit and input buffers with four positions, was used to validate the timing and energy models. The Hermes VHDL description was synthesized to an ASIC standard cell library. The library also supplies energy values for the cells, which are used to extract the energy parameters.

The synthesis result is a logic gate netlist. This netlist is associated to a customized VHDL library, which enables fast and accurate energy consumption and timing estimations. A testbench applies both random and typical traffic to the netlist and the results achieved by VHDL simulation are compared to those obtained from high-level mapping tool. Our experiments showed average errors below 30.5% and 14% for energy consumption and execution time estimations, respectively.

V. Experimental Setup

This section presents the methods used to generate the combination of faulty tiles, called fault scenarios. The first method is exhaustive used for small NoCs and the second method is the statistical method used for bigger NoCs. Latter, we present the application classes evaluated in this paper.

A. Exhaustive Fault Generation Method

Faulty tiles are *exhaustively generated* for all combinations of faulty tile locations, assuming a system with 1 to 3 faulty tiles. Thus, Equation 11 defines the total number of faults injected as the sum of all 1, 2, to *nfaults* faults combination in $x \times y$ tiles. For instance, a 3×4 mesh NoC requires **298** fault scenarios (12 single faults, 66 double faults, and 220 triple faults).

$$scens(x \times y, \text{nfaults}) = \binom{x \times y}{1} + \binom{x \times y}{2} + \cdots + \binom{x \times y}{\text{nfaults}} \qquad (11)$$

B. Statistical Fault Generation Method

The exhaustive fault generation method is precise; however, it might not be possible to perform exhaustive fault simulation due to the long CPU time. The main reason is that the total number of executions required to perform exhaustive fault simulation, defined in Equation 11, grows exponentially with the NoC size ($x \times y$) and the max number of simultaneous faults (*nfaults*). Moreover, the CPU time of a single execution of the task mapping tool grows with the NoC size.

For instance, assuming a 3 x 4 mesh NoC with up to 3 simultaneous faults requires 298 task mapping executions (about 3 minutes of CPU) to perform exhaustive fault simulation. However, a bigger NoCs like a 5 x 5 mesh with up to 3 faults requires 2625 executions in about 60 hours of CPU time. The same 5 x 5 mesh NoC with up to 4 simultaneous faults requires 15275 executions, which we estimate that would require about 14 days of CPU use.

Even with the economical motivation of spare tiles is appealing; it might be unfeasible to perform an exhaustive fault simulation since the CPU time becomes an issue for bigger NoCs with multiple faults. This section presents a statistical approach, called *sample size estimation* [25], used to *determine the minimal number of fault scenarios required to have satisfactory results* - near to the ones achieved by exhaustive approach. This way, CPU time can be drastically reduced, while the results are still accurate. Moreover, this method enables trading off CPU time and result accuracy.

Before executing the sample size estimation, a pilot simulation is performed with a sample of small size. A *sample* represents a set of executions of the task mapping tool, where each execution assumes that the faulty tiles were randomly selected. Each execution of this pilot results in a different mapping with different energy consumptions and execution times. If the energy consumption is the value to be estimated, then this pilot gives the *population's estimated standard deviation* S of energy consumed in the presence of faulty tiles randomly located. The *population* in this context represents the entire combination of fault scenarios, as determined in Equation 11.

The goal of the sample size estimation is to estimate the population average (μ), *i.e.* the average energy consumption of the entire population of fault scenarios. The Equation 12 is typically used for this purpose, where s is the estimated standard deviation of the sample. $(x - \mu)$ is the difference of the estimated sample average (x) and μ, which represents the acceptable error between the sample and the population. $t_{\alpha,df}$ is the value from student's t-distribution table [25], where $(1 - \alpha)$ is the confidence level and df is the degree of freedom, defined as $df = n - 1$.

$$n = \frac{s^2}{(x-\mu)^2} \times (t_{\alpha,df})^2 \qquad (12)$$

Since n is unknown, one can select an initial value of n to obtain $t_{\alpha,df}$. This value is used in Equation 12 to find a new n and a new $t_{\alpha,df}$. This calculation is performed iteratively until the value of n stabilizes. The stable value of n is the minimal sample size required to estimate the population average μ with the expected accuracy of results.

C. System Application

We explore several parallel applications with distinct features aiming to determine what kind of application increases the overhead in energy and execution time in the presence of faulty tiles. A synthetic application generator, detailed in [22], is used to create random CDCGs.

This synthetic application generator can build several application classes by varying parameters such as: (i) *number of processors*, which allows to investigate some target architecture dimensions; (ii) *number of graph levels*, which allows to specify the number of dependent communications an application has; (iii) *dependence degree* that defines the probability of a vertex has more than one dependence, keeping in mind that dependent communications can´t concur for NoC resources; (iv) *probability of end meeting* that defines if a vertex will have dependences or is a final communication; (v) *computation time* that is the period, associated to each source task, between all dependences are solved and the communication of the source task starts; (vi) *communication volume* that contains the quantity of bytes transmitted in each communication; and (vii) *parallel communications*, which describes the minimum quantity of parallel communications an application have.

For instance, varying the relation between computation time and communication volume the application may change from IO-bounded to CPU-bounded; varying the relation between number of graph level and dependence degree the applications may be dataflow or concurrent.

We built 39 synthetic applications, which enables to explore applications classified as (i) IO or CPU bounded; (ii) dataflow with different levels of parallelism; (iii) strongly parallel or sequential applications with different levels of concurrence by the communication architecture.

VI. Experimental Results

This section evaluates (*i*) the application execution time under exhaustive faulty scenarios, (*ii*) the average energy consumption under exhaustive faulty scenarios, (*iii*) the proposed statistical fault generation method used to estimate energy consumption, comparing it to the exhaustive method.

A. Evaluating the Application Execution Time

All application classes haves been evaluated in terms of execution time using the exhaustive fault generation method.

The result is that, *independently of the application class, the execution time is not affect by the presence of faulty tiles*. On average, the variation of execution time between the fault-free chip and the chips with up to 3 faulty tiles is close to 0%.

The reason lies on the timing model, presented in Section IV.B, more specifically in Equation (3). The total application time consist of computation time plus the communication time. The communication time consist of the routing delay, which depends on the distance between the communication elements, plus the packet delay, which depends on the packet size.

If the computation time is much greater than the communication time, then the task mapping has very small influence on the application execution time. Even if both computation and communication times are equivalent, if the packet size is big (hundreds of flits) the routing delay has a very small impact on the communication time (since the NoC works as a pipeline), thus also a small impact on the application execution time.

Since in typical scenarios an application has more computation than communication and applications use packets with hundreds of flits, then the impact of the routing delay on the overall application execution time almost is negligible. This claim can be demonstrated with the following example.

Let us assume a given application on a 3x4 mesh NoC, whose normal behavior is to have more computation than communication. This application is modified such that it has three variations: low communication (packets of one flit), low computation (CPU time of 1 clock cycle), and both communication and computation are low. Exhaustive fault generation is performed for these cases generating the Figure 4, which is the difference between the average execution time of the population with faulty chips and the execution time of the fault-free chip.

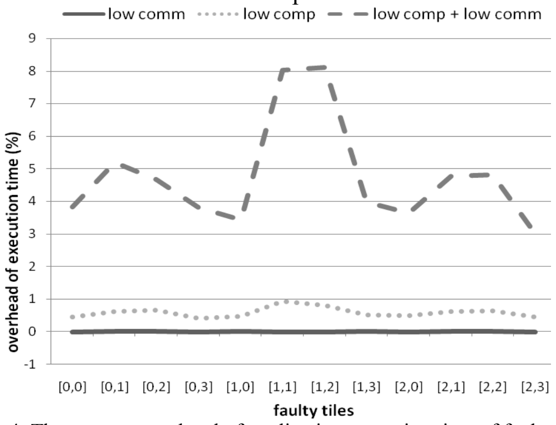


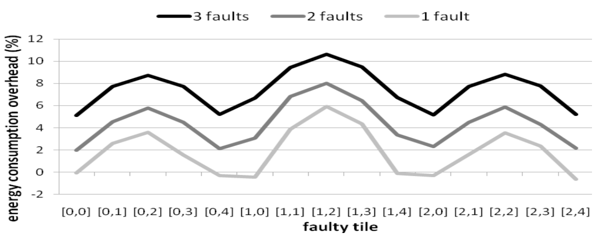
Figure 4. The average overhead of application execution time of faulty chips.

This figure demonstrates that faulty tiles have a significant influence on the chip execution time only if the both the computation and the communication are low, which is not the typical situation. Most actual applications typically have bigger packet sizes and more computation than communication.

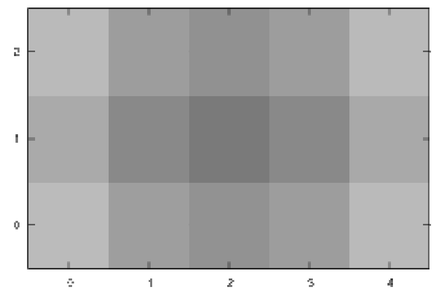
B. Evaluating the Energy Consumption

The energy consumption is evaluated for each class of application described in Section V.C. The result is that, in spite of the application class, *only the proportion of good tiles per faulty tiles affects the energy consumption*. For instance, a chip with 15 tiles where two of them are faulty consumes more energy than the same chip with only one faulty tile. These results are illustrated in Figure 5 for a 3x5 mesh and an application with 12 tasks and 3 spare tiles. The average impact of a faulty tile on energy consumption is worse in the center of the NoC and it increases if there are more faulty tiles in the chip (Figure 5(a)). This impact gradually decreases as the distance from the center tiles increases (Figure 5(b)).

However, if we map the same application on a 4x4 mesh NoC, then there are 12 tasks and 4 spare tiles. Figure 6 compares the energy profile of this application of a 3x5 against a 4x4 mesh NoC assuming three faults in each of them. It can be observed that the energy overhead in a 4x4 is lower. The *reason is the proportion of good and faulty tiles*. In a 3x5 with 3 faults the proportion is 15/3 while in a 4x4 it is 16/3. This extra tile of 4x4 gives more freedom to the task mapping tool to determine a good scheduling, improving the effect of self-similarity (Section IV.A), resulting in a better task mapping.



(a)



(b)

Figure 5. The average energy consumption overhead of faulty chips.

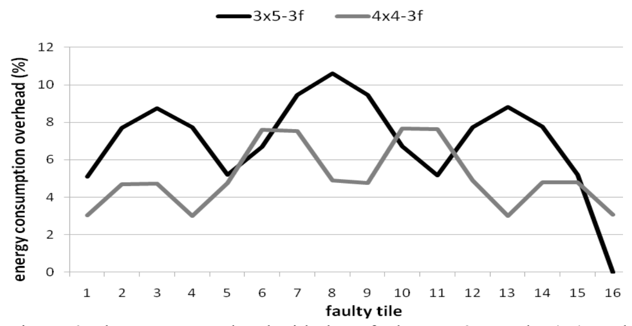


Figure 6. The energy overhead with three faults on a 3x5 and a 4x4 mesh.

C. Evaluating the Statistical Fault Generation Method

This section demonstrates the fault generation method proposed in Section V.B. For this experiment, we assume that a small NoC is used, such as 3x4 mesh NoC, because the total CPU time for both statistical and exhaustive fault generation methods is not too high. An application with 9 tasks is used for this experiment, even though all other applications presented very similar results. Let us assume that the goal of this experiment is to estimate the average energy overhead when a fault hit a given tile, considering scenarios with 3 simultaneous faults.

First, the exhaustive method is executed, running all combinations of 3 faults in 12 tiles, *i.e. scens*(3 x 4, 3) = 298 (Eq. 11) fault scenarios. It took about 3 minutes of CPU time to execute them. These results are considered the target results, i.e. the results we want to achieve with the statistical method.

The second step is to execute a pilot experiment with small number of randomly selected fault scenarios per router. This pilot experiment is used solely to extract the standard deviation of the energy consumed by the chips with three random faulty tiles. The estimated standard deviation is 3.9% of deviation on energy consumption.

The proposed approach of sample size estimation is executed assuming two situations: (*i*) standard deviation of 3.9, confidence interval of 95%, and maximum error of 4%; and (*ii*) standard deviation of 3.9, confidence interval of 98%, and maximum error of 2%. The estimated sample size for each situation is 8 and 23, respectively. It means that each tile must be in at least 8 or 23 fault scenarios. For now on, the first situation is called *sample8* and the second is called *sample23*. TABLE 1 presents the obtained results in terms of CPU time, total number of scenarios and the maximum error observed for each tile.

TABLE 1. RESULTS FOR THE STATISTICAL FAULTS GENERATION METHOD.

	CPU time (s)	# scenarios	max obs. error (%)
Exhaustive	192	220	-
Sample8	27	39	2.4
Sample23	63	101	0.8

Figure 7 illustrates the three situations and their respective heat charts, representing the energy overhead when a fault is found at each tile. Each square represent the average energy consumption for each tile.

It can be observed that the exhaustive method produce the expected results (the energy is gradually reducing from the center to the borders). The sample23 produces almost the same results as the exhaustive method, with small error but with much less CPU time. The sample8 produce large errors, indicating that the sample size is not sufficient to estimate accurately the energy overhead for each router.

Even if the exhaustive results are not available, it is still possible to check the accuracy of the sample by visually analyzing the heat chart demonstrated in Figure 7. For instance, the expected appearance of a good heat chart is like the exhaustive test set, even if we assume NoCs of different sizes and different applications. Note that the heat chart for *sample8* deviates from the expected appearance, indicating that one should increase the sample size, if it is possible, to increase the accuracy of the results.

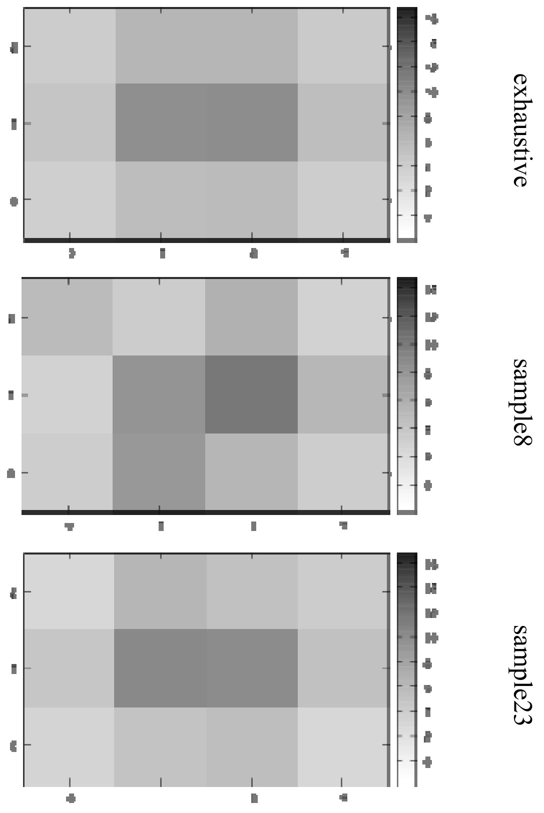


Figure 7. Visual analysis of the statistical fault generation method.

Figure 8 overlaps the average results for the three situations. By comparing the exhaustive with the other test sets, it can be seen that the biggest error for sample8, located in the tile [2, 1], is 2.4% (see 1), which is below the maximum error stipulated to this set of experiments (4%). The biggest errors for sample23, located in the tiles [1, 1] and [2, 0] (see 2), are around 0.8%, which is below the maximum error stipulated to this set of experiments (2%).

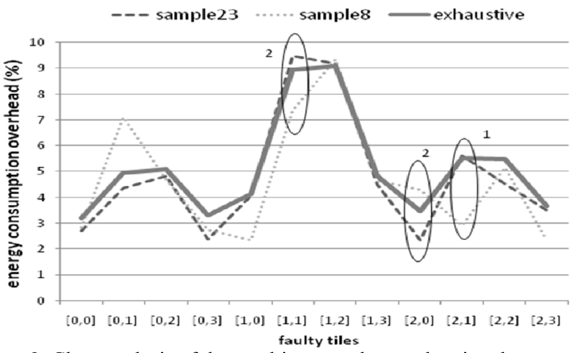


Figure 8. Close analysis of the resulting error by overlapping the average results for exhaustive, sample8, and sample23.

The example presented in this section demonstrates that the proposed fault generation approach enables to trade-off CPU time and result accuracy by selecting different values of difference (***x* - *μ***) and confidence level (***1* - *α***).

VII. Final Remarks

Previous papers have demonstrated that the use of spare tiles can significantly improve yield and reduce the manufacturing cost of NoC-based MPSoCs. The tool presented in this paper determines task mapping for NoC-based MPSoCs with faulty tiles, minimizing the energy consumption and the application execution time. This way, these defective chips can still execute the application, perhaps with some performance degradation, but at least it can be sold to a lower-end market, for example.

This paper evaluates energy consumption and application execution time of faulty chips compared to fault-free chips. We evaluated several different classes of applications to check if there was any particular application feature that could affect the energy consumption or application execution time under faulty tiles. These results show that the spare tile approach has small impact on energy consumption and this impact can be even smaller if the proportion of good and faulty is higher. The existence of faulty tiles on the chip has, on average, no significant influence on the application execution time. Based on these results, we conclude that the spare tile approach can increase yield and cost with small penalties on the application requirements.

Finally, this paper also proposed a statistical fault generation approach targeting very large MPSoCs. This approach demonstrates that a small sample of fault scenarios is sufficient to have a reasonably accurate estimation of energy consumption and it enables trading of CPU time and result accuracy.

VIII. Acknowledgment

Alexandre is supported by postdoctoral scholarships from Capes-PNPD and FAPERGS-ARD, grants number 02388/09-0 and 10/0701-2, respectively. Fernando Moraes is supported by CNPq and FAPERGS, projects 301599/2009-2 and 10/0814-9, respectively. Cesar Marcon and Marcelo Lubaszewski are partially supported by CNPq scholarships, grants number 308924/2008-8 and 478200/2008-0, respectively.

IX. References

[1] Wolf, W.; Jerraya, A. A.; Martin G. **Multiprocessor system-on-chip (MPSoC) technology**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10), pp. 1701-1713, 2008.

[2] Kahng, A.; et al. **ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration**. *DATE*, pp. 423-428, 2009.

[3] Lee, S. E. et al. **A high level power model for network-on-chip (NoC) router**. *Computers & Electrical Engineering*, 35(6), 2009.

[4] Refan, F. et al. **Reliability in application specific mesh-based NoC architectures**. *IEEE International On-Line Testing Symposium*, pp. 207-212, 2008.

[5] Shamshiri, S.; Cheng, K-T. **Yield and Cost Analysis of a Reliable NoC**. *VLSI Test Symposium*, pp. 173-178, 2009.

[6] Carara E. A. et al. **HeMPS - a framework for NoC-based MPSoC generation**. *ISCAS*, pp. 1345–1348, 2009.

[7] Moraes, F. et. al. **HERMES: an infrastructure for low area overhead packet-switching networks on Chip**. *Integration, the VLSI Journal*, 38(1), pp. 69-93, 2004.

[8] Bertozzi, D.; Benini, L.; De Micheli, G. **Error control schemes for on-chip communication links: the energy-reliability tradeoff**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(6), pp. 818-831, 2005.

[9] Ejlali, A. et al. **Performability/energy tradeoff in error-control schemes for on-chip networks**. *IEEE Transactions on Very Large Scale Integration Systems*, 18(1), pp. 1-14, 2010.

[10] Zhang, Z.; Greiner, A.; Taktak, S. **A reconfigurable routing algorithm for a fault-tolerant 2D-mesh network-on-chip**. DAC, pp. 441-446, 2008.

[11] Manolache, S.; Eles, P.; Peng, Z. **Fault and energy-aware communication mapping with guaranteed latency for applications implemented on NoC**, DAC, pp. 266-269, 2005.

[12] Hu, J.; Marculescu, R.. **Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints**. *DATE*, pp. 234-239, 2004.

[13] Lei, T.; Kumar, S. **A two-step genetic algorithm for mapping task graphs to a network on chip architecture**. *Euromicro Symposium on Digital System Design*, pp. 180-187, 2003.

[14] Murali, S. et al. **Mapping and configuration methods for multi-use-case networks on chips**. *ASP-DAC*, pp. 146-151, 2006.

[15] Lee, C. et al. **A task remapping technique for reliable multi-core embedded systems**. *CODES/ISSS*, pp. 307-316, 2010.

[16] Ababei, C.; Katti, R. **Achieving network on chip fault tolerance by adaptive remapping**. *International Symposium on Parallel & Distributed Processing*, pp. 1-4, 2009.

[17] Tornero, R. et al; **A multi-objective strategy for concurrent mapping and routing in networks on chip**. *International Symposium on Parallel & Distributed Processing*, pp. 1-8, 2009.

[18] Choudhury, A. et al. **Yield enhancement by robust application-specific mapping on network-on-chips**. *NoCArc*, pp. 37-42, 2009.

[19] Huang, L. et al. **Lifetime reliability-aware task allocation and scheduling for MPSoC platforms.** *DATE*, pp. 51-56, 2009.

[20] Huang, L; Xu, Q. **Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint**. *DATE*, pp. 1584-1589, 2010.

[21] Huang, L; Xu, Q. **AgeSim: A simulation framework for evaluating the lifetime reliability of processor-based SoCs**, *DATE*, pp. 51-56, 2010.

[22] Marcon, C. et al. **CAFES: a framework for intrachip application modeling and communication architecture design**. Journal of Parallel and Distributed Computing, 71(5), pp. 714-728, 2011.

[23] Mandelbrot, B. **How long is the coast of britain? statistical self-similarity and fractional dimension**. *Science*. 156(3775) pp. 636-638, 1967.

[24] Ghadiry, M.; Nadi, M.; Rahmati, D. **New approach to calculate energy on NoC**. *International Conference on Computer and Communication Engineering*, pp. 1098-1104, 2008.

[25] Hill, T.; Lewicki, P. **Statistics: methods and applications: a comprehensive reference for science, industry, and data mining**. *StaSoft,* 832p., 2006.