Online Task Remapping Strategies for Fault-tolerant Network-on-Chip Multiprocessors

Onur Derin Deniz Kabakci Leandro Fiorin
ALaRI - Faculty of Informatics - University of Lugano
Via Buffi 13, 6904, Lugano, Switzerland
name.surname@usi.ch

ABSTRACT

As CMOS technology scales down into the deep-submicron domain, the aspects of fault tolerance in complex Networks-on-Chip (NoCs) architectures are assuming an increasing relevance. Task remapping is a software based solution for dealing with permanent failures in processing elements in the NoC. In this work, we formulate the optimal task mapping problem for mesh-based NoC multiprocessors with deterministic routing as an integer linear programming (ILP) problem with the objective of minimizing the communication traffic in the system and the total execution time of the application. We find the optimal mappings at design time for all scenarios where single-faults occur in the processing nodes. We propose heuristics for the online task remapping problem and compare their performance with the optimal solutions.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Algorithms, Performance, Reliability, Theory

Keywords

adaptivity, fault-tolerance, Kahn Process Networks, Networks-on-Chip, mapping

1. INTRODUCTION

As CMOS technology scales down into the deep-submicron domain, the aspects of fault tolerance in complex Networks-on-Chip (NoCs) architectures are assuming an increasing relevance. In fact, devices and interconnect are subjected to new types of malfunctions and failures that are often hardly predictable and avoidable with current design methodologies [16]. Fault tolerant approaches are therefore necessary to overcome these limitations, and new methodologies based either on architectural or software solutions should be explored. Faults in NoCs may affect both the communication system and the cores, and different solutions in the literature have been proposed to address these two specific classes of malfunctioning.

In this work, we focus on the use of software based solutions for tolerating permanent failures in processing elements, and allowing a graceful degradation of the system performance by remapping online tasks running on faulty processors. Tasks reallocation represents an alternative solution to the classical use of resource redundancy, which exploits the intrinsic availability of spare computation resources available in modern Multiprocessor Systems-on-Chip platforms. Moreover, it represents an obliged choice in particular in embedded systems, constrained in terms of number of computation resources available.

Our paper presents two main contributions. The first one is a method for finding an optimal solution to mapping tasks onto heterogeneous NoC multiprocessor systems. In most of related works [14, 11], the problem of optimal task mapping has been addressed in two phases. The first phase addresses the partitioning problem, which deals with the selection of the IP for implementing the tasks of the application. Core mapping constitutes the second phase, where the selected IPs are mapped on the tiles of the NoC. The partitioning deals with optimization of the computation, whereas core mapping deals with optimizing communication. Partitioning problem starts with the task graph and a list of IPs; and results in a core communication graph (CCG). Core mapping problem starts with the CCG and results in a mapping of cores to tiles. Our formulation combines the two steps, and starting from a task graph provides as output the mapping of tasks onto tiles. Therefore both computation and communication is optimized in a single step. We adopt the platform-based design paradigm, where the NoC architectural platform with its pre-selected and pre-placed IPs is already given. Goal of the methodology is to execute a given application on this platform optimizing both computation and communication.

To achieve this goal, we propose an ILP formulation for the problem. The same methodology is also applicable to the core mapping problem, by interpreting the task graph as a CCG and by considering an additional constraint which imposes to map at most one task onto each tile. To the best of our knowledge, the use of ILP formulation for finding

NOCS '11, May 1-4, 2011 Pittsburgh, PA, USA

an optimal solution to the task mapping problem onto heterogeneous NoC multiprocessors has not been presented in previous works. There have been several works that proposed as possible solution to the optimization problem the use of heuristics [8, 18, 9, 15, 23], evolutionary algorithms [2, 25, 11, 3, 22, 7] and a mix of both [19, 17]. However, the usual approach followed is to compare the performance of the proposed solutions with each other, or with a solution found by applying simulated annealing. An ILP formulation guarantees to find an optimal solution to the problem; however, since the ILP solution is not scalable, and the needed execution time will increase significantly with the dimension of the NoC, we acknowledge that heuristics and evolutionary algorithms are of significant value. Nevertheless, an optimal solution to the problem will also serve a good deed in comparing the performances of heuristics-based solutions. In fact, results of approaches based on evolutionary algorithms are not guaranteed to be optimal whereas ILP results are.

ILP-based solutions to similar problems have been proposed in the case of contention-aware application mapping on NoC [5]; floor-planning and topology generation for NoC [20]; task mapping and scheduling on multi-core architectures [24]; task mapping on shared memory bus-based heterogeneous MPSoCs [6]. It is possible to classify the previous works in terms of the tackled problem (core mapping, task mapping, partitioning, allocation, scheduling, routing, topology generation); optimization goals (execution time, delay, communication, power, robustness, contention, flexibility); optimization techniques (heuristics, evolutionary algorithms, exact solutions); architectural platform (fixed/free NoC topology, fixed/free routing). Following such criteria, our solution can be classified as an ILP solution to the task mapping problem with minimization of total execution time, and minimization of total communication, on mesh-based NoC architectures with fixed routing algorithms.

Two of the most related works that target the application mapping problem are [21] and [12]. They both apply an optimization based on the use of genetic algorithms; the former optimizes the total execution time and communication load; the latter optimizes the throughput, area and flexibility. With respect to these works, our approach focuses on platforms in which each NoC tile has one computational core (either programmable or non-programmable).

In this work, we refer to a mesh-based NoC multiprocessor architecture. Our analytical model for calculating the total execution time of the application is valid for Kahn Process Networks (KPN). Thus our task graphs are restricted to KPN task graphs making our solution applicable to mostly streaming applications. Our formulation is valid for any deterministic routing scheme. Deterministic routing implies that communication binding is implied by the task mapping and that the task mapping is the only degree of freedom.

Our second main contribution is an online solution to the task remapping problem in presence of run-time faults. We propose heuristics and make a comparison of performance degradations between heuristics and the optimal solutions found by using the ILP formulation for various fault scenarios. Regarding related work in this research direction, in [1] authors address the core remapping problem in NoCs.In that case, the only concern for remapping is the minimization of the communication load. Whereas, in our case, we move individual tasks on a core to, possibly, different cores. Their remapping strategy can be imported to our context as the center of gravity (CoG) technique described in section 4. Our results reveal that CoG does not perform well for the computation objective.

The remainder of this paper is organized as follows. Section 2 presents the ILP formulation for the mapping problem. In Section 3, we apply the proposed methodology to an MPEG2 decoder case study, showing the obtained results. In Section 4, we propose and evaluate a set of heuristics for online task remapping using the MPEG2 decoder case study. Finally, in Section 5, we discuss conclusions and future work.

2. ILP FORMULATION OF THE PROBLEM

- A *task graph* $g_t = (V_t, E_t)$ is composed of tasks $t \in V_t$ and data dependencies $e \in E_t \subseteq V_t \times V_t$.
- An *architecture graph* $g_a = (V_a, E_a)$ is composed of processing nodes $n \in V_a$ and bidirectional communication links $l \in E_a \subseteq V_a \times V_a$.
- A *task mapping* $\beta_t : V_t \rightarrow V_a$ is an assignment of tasks $t \in V_t$ to nodes $n \in V_a$.
- A *communication binding* $\beta_c : E_t \rightarrow E_a^i$ is an assignment of data dependencies $e \in E_t$ to paths of length i in the architecture graph g_a. A *path* p of length i is given by i-tuple $p = (l_1, l_2, ..., l_i)$.
- $path : (E_a, E_a) \rightarrow E_a^i$ is a function that implements a deterministic routing algorithm and returns a path between two given nodes. *Path set* P is the set of paths between all node pairs:

  $$P = \{p_k : p_k = path(n_i, n_j), \forall n_i, n_j \in V_a \wedge n_i \neq n_j\}$$

  Initial and final nodes of a path can be obtained by *source* and *sink* functions.

  $$p_k = path(n_i, n_j) \Rightarrow source(p_k) = n_i \wedge sink(p_k) = n_j$$

- The task graph can be annotated with demand values where *demand* d_i on a data dependency $e_i \in E_t$, denotes the required bandwidth between the two tasks. Demand values are application specific and can be calculated by profiling the application with a test input.
- The architecture graph can be annotated with capacity values where *capacity* on an architectural link $l_i \in E_a$, c_i, denotes the maximum bandwidth of the communication link between two architectural nodes.
- Core type set C consists of core types C_i and lists the types of cores available in a given NoC platform.

We would like to minimize the total network traffic and the computation time.

2.1 Minimization of the communication cost

In order to formulate the problem, we define several incidence matrices, namely decision variables X^{NT}, Y^{PE}; and parameters M^{TE}, M^{NP} and M^{PL}.

X^{NT} is an incidence matrix of size $|V_a| \times |V_t|$ that denotes the mapping of tasks onto the nodes and it consists of the main decision variables of the problem.

$$X_{ij}^{NT} = \begin{cases} 1, & \text{if } t_j \in V_t \text{ is bound onto node } n_i \in V_a \\ 0, & \text{otherwise} \end{cases}$$

Y^{PE} is an incidence matrix of size $|P| \times |E_t|$ that denotes which path realizes which data dependency. Y^{PE} depends on the task mapping, hence it constitutes the second set of our decision variables.

$$Y^{PE}_{ij} = \begin{cases} 1, & \text{if } e_j \in E_t \text{ is mapped to } p_i \in P \\ 0, & \text{otherwise} \end{cases}$$

M^{TE} is an oriented incidence matrix of size $|V_t| \times |E_t|$ that relates the tasks to the data dependencies. For a given task graph, M^{TE} is known.

$$M^{TE}_{ij} = \begin{cases} 1, & \text{if } \exists t_k, e_j = (t_i, t_k) \in E_t \\ -1, & \text{if } \exists t_k, e_j = (t_k, t_i) \in E_t \\ 0, & \text{otherwise} \end{cases}$$

M^{NP} is an oriented incidence matrix of size $|V_a| \times |P|$ that denotes the relation between the paths and the nodes that the path connects. For a given routing algorithm and architecture graph, M^{NP} is known.

$$M^{NP}_{ij} = \begin{cases} 1, & \text{if } source(p_j) = n_i \\ -1, & \text{if } sink(p_j) = n_i \\ 0, & \text{otherwise} \end{cases}$$

M^{PL} is an incidence matrix of size $|P| \times |E_a|$ that denotes the relation between all paths resulting from a given deterministic routing algorithm and the links that make up the path. For a given routing algorithm and architecture graph, M^{PL} is known.

$$M^{PL}_{ij} = \begin{cases} 1, & \text{if } l_j \in p_i \\ 0, & \text{otherwise} \end{cases}$$

Constraint 1 *(routing)*: We have derived the following linear equation that constrains the task mapping and the communication binding with each other. Such a constraint arises from the routing algorithm implemented in the NoC.

$$X^{NT} M^{TE} = M^{NP} Y^{PE} \tag{1}$$

Constraint 2 *(task mapping)*: A task can be mapped exactly on one node.

$$X^{TN} \mathbf{1}_{|V_a|} = \mathbf{1}_{|V_t|} \tag{2}$$

where $\mathbf{1}_m$ is a matrix of size $m \times 1$ with all elements equal to 1. It is to be noted that $X^{TN} = (X^{NT})^T$. Similar relation holds for all defined matrices.

Constraint 3 *(communication mapping)*: A data dependency can be mapped at most on one path.

$$Y^{EP} \mathbf{1}_{|P|} \leq \mathbf{1}_{|E_t|} \tag{3}$$

Constraint 4 *(capacity)*: Total bandwidth demand on a link l_j should not exceed the capacity of the link c_j.

$$M^{LP} Y^{PE} d \leq c \tag{4}$$

Objective 1 *(communication cost)*: The total traffic on the links can be calculated as the sum of all demands d_i on the links of the paths that arise according to a given mapping with the following equation.

$$\text{min: } d^T \, Y^{EP} M^{PL} \, \mathbf{1}_{|E_a|} \tag{5}$$

This is a static model that has also been used in [18] and disregards the congestion on the links. However at low load conditions, it is argued that it is a good approximation.

Note that the communication cost takes into account the inter-tile communication done over the NoC between tasks and not the intra-tile communication when communicating tasks are mapped onto the same node. The latter is usually much faster compared to the former.

Therefore the objective for communication is the minimization of the total traffic (Equation 5) subject to routing algorithm constraints (Equation 1), mapping constraints (Equation 2, 3) and capacity constraints (Equation 4). Since the equations are linear, this problem can be solved with an integer linear programming (ILP) solver.

Given our analytical cost model, it is obvious that when communication cost is taken as the only objective, the resulting mapping will always be that all tasks are mapped on a single node. However this will reflect badly on the computation time due to non-parallelism. Therefore we introduce a conflicting second objective that favors tasks to be placed on separate nodes.

2.2 Minimization of the total computation time

In order to formulate the problem we define additional parameters in matrix form, namely M^{TC}_{cap}, T^{TC}_{cap} and M^{NC}.

M^{TC}_{cap} is an incidence matrix of size $|V_t| \times |C|$ that denotes which core types are capable of realizing which tasks. Programmable cores would be capable of realizing different kinds of task functionalities, whereas non-programmable cores would have dedicated functions.

$$M^{TC}_{cap\,ij} = \begin{cases} 1, & \text{if } C_j \in C \text{ is capable of realizing task } t_i \in V_t \\ 0, & \text{otherwise} \end{cases}$$

T^{TC}_{cap} is a matrix of size $|V_t| \times |C|$ that denotes the completion times of all tasks on all core types for a test input. This value is obtained multiplying the number of times the task body is executed by the time it takes to process at each firing. Given an application and architecture, this matrix can be obtained by offline profiling.

$$T^{TC}_{cap\,ij} = \begin{cases} \text{completion time of } t_i \text{ on } C_j, & \text{if } M^{TC}_{cap\,ij} = 1 \\ 0, & \text{if } M^{TC}_{cap\,ij} = 0 \end{cases}$$

M^{NC} is an incidence matrix of size $|V_a| \times |C|$ that denotes the core type of the architectural nodes. Given an architecture, M^{NC} is known.

$$M^{NC}_{ij} = \begin{cases} 1, & \text{if } n_i \in V_a \text{ is of core type } C_j \in C \\ 0, & \text{otherwise} \end{cases}$$

M^{TC} is an incidence matrix of size $|V_t| \times |C|$ that denotes the actual mapping of tasks on core types. Given a task to node mapping matrix, X^{TN}, it can be calculated as

$$M^{TC} = X^{TN} M^{NC}$$

T^T is a matrix of size $|V_t| \times 1$ that denotes the completion time of the task on the core type that it is mapped onto. It can be calculated as

$$T^T = (M^{TC} \; . \; T^{TC}_{cap}) \; \mathbf{1}_{|C|}$$

where the '.' operator represents element-wise matrix multiplication.

T^N is a vector of size $|V_a| \times 1$. T^N_i denotes the sum of execution times of tasks that are mapped on the same node, n_i. It can be calculated as

$$T^N = X^{NT} T^T$$

Constraint 5 *(capability)*: All tasks should be mapped on cores that are capable of implementing those tasks.

$$M^{TC} = X^{TN} M^{NC} \leq M^{TC}_{cap} \tag{6}$$

Objective 2 *(total execution time)*: We calculate the total computation time of the application by finding the maximum of the sum of the execution times of tasks mapped on the same core.

$$\text{min: } max(T^N) = max(X^{NT}(((X^{TN} M^{NC}) \ . \ T^{TC}_{cap}) \ \mathbf{1}_{|C|})) \tag{7}$$

where max is a function that returns the maximum value in a given vector.

This is a static model that has also been used in [21] and disregards the context switching times. However it is argued that this model has a reasonable accuracy for typical streaming applications.

The objective for computation is the minimization of the total execution time (Equation 7) subject to capability constraints (Equation 6). The objective function in Equation 7 is not linear due to the multiplication $X^{NT}X^{TN}$ and the max function. However, there are linearization techniques that transform these equations to linear counterparts by introducing new variables.

Linearization of the product of binary decision variables in $X^{NT}X^{TN}$ by introducing new binary decision variables:

$$x_{ijkl} = x_{ij} * x_{kl} \Rightarrow \begin{cases} x_{ijkl} \leq x_{ij} \\ x_{ijkl} \leq x_{kl} \\ x_{ijkl} \geq x_{ij} + x_{kl} - 1 \end{cases}$$

Linearization of $max()$ can be done by introducing a new variable:

$min : max(x, y, z) \Rightarrow min : t$ subject to $t \geq x$, $t \geq y$, $t \geq z$

2.3 Multi-objective ILP optimization

We have defined two ILP problems that optimize two objectives separately. What we actually need is the multi-objective optimization of the combined problem that should result in a Pareto curve representing optimal solutions with different trade-offs for the two objectives. This is done by employing the ε-constraint method [4]. This method relies on adding one of the objectives as a constraint by requiring it to be smaller than a chosen threshold. By solving the ILP problem several times for different values of the threshold and for a single objective, we obtain a Pareto curve. It is also possible to adjust the density of Pareto curve by adjusting the intervals of the threshold range.

It is worth-noting that the solutions found by the multi-objective ILP optimization are absolute optima unlike what would be obtained by evolutionary algorithms. However we acknowledge the fact that the time required to solve the ILP problem will be multiplied by the desired number of Pareto points.

3. CASE STUDY

We use the MPEG-2 decoder task graph shown in Figure 1 and offline profiling taken from [21]; and the XY-routing-based NoC architecture in Figure 2 to illustrate the problem formulation better. The throughput of the links of the NoC is 100 MBps. The video is 15 seconds long with a resolution of 704×576 pixels and the frame rate is 25 frames per second.

M^{TE} can be obtained from the task graph. M^{NP} and M^{PL} can be obtained from the given architecture. We avoid writing them here due to space constraints. T^{CT}_{cap} and d are given in Table 1 and 2 respectively. Other parameters are listed as follows:

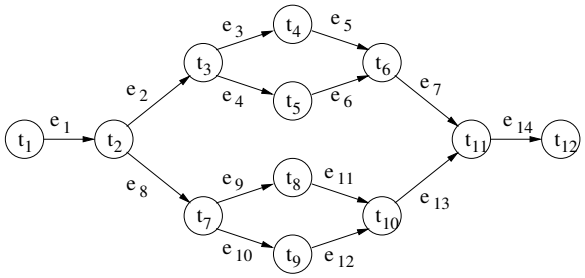


Figure 1: An MPEG-2 encoder task graph with 12 tasks

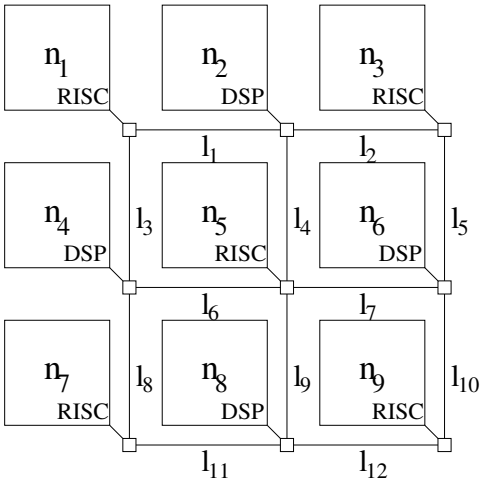


Figure 2: A 3x3 mesh-based NoC with RISC and DSP processors

$$c_i = 100 \text{ MBps}, \ 1 \leq i \leq |E_a|$$

$$C = \{C_1, C_2\} = \{\text{RISC}, \text{DSP}\}$$

$$M^{TC}_{cap} = \mathbf{1}_{12\times 2}$$

3.1 Results

The case study described in section 3 has been solved using the IBM ILOG CPLEX optimizer. Figure 3 shows the Pareto curve obtained. Considering that it was a 15 seconds long video clip, the solutions that satisfy the frame rate requirement are those that have computation time less than 15 seconds. One can choose the mapping among the rest by trading off computation time and communication load.

In order to have case studies with more number of tasks, we have solved the optimal mapping problem of 2 and 3 MPEG2 decoders giving us two more task graphs with 24 and 36 tasks. The Pareto curves for these cases are also plotted in Figure 3. The time for the ILP solver to obtain these solutions was in average 0.78 sec, 27.87 sec and 1740 sec per Pareto point for the cases with 12, 24 and 36 tasks, respectively.

Table 1: Execution times (in seconds) of tasks on the available core types (T_{cap}^{CT})

Core type	Tasks											
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
RISC	0.13	6.68	0.06	2.00	2.00	0.05	0.06	2.00	2.00	0.05	12.33	0.18
DSP	0.20	8.52	0.04	1.25	1.25	0.04	0.04	1.25	1.25	0.04	8.51	0.30

Table 2: Bandwidth demands (in MBps) of edges (d)

d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}
1.0	34.6	28.1	28.1	28.1	28.1	65.0	34.6	28.1	28.1	28.1	28.1	65.0	15.2

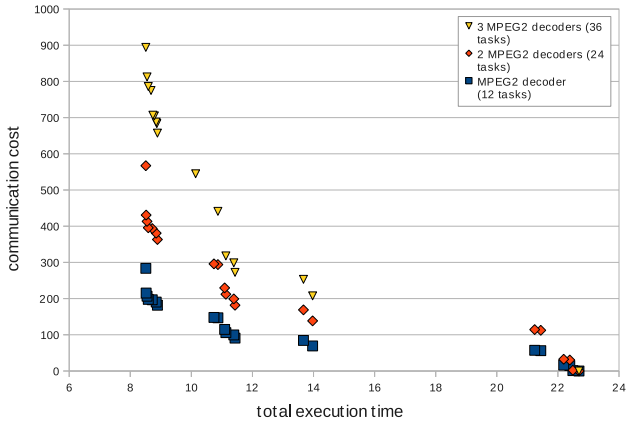


Figure 3: Pareto curves for optimal mapping of three applications with 12, 24 and 36 tasks

4. ONLINE TASK REMAPPING

Our overall goal is enabling the execution of KPN applications on NoC platforms in a fault tolerant manner. Self-testing and self-checking are two techniques to detect and, in the case of the latter, mask faults. After the faulty unit is detected, it is followed by a reconfiguration phase. This phase enables isolation of the fault and continuity of operation, possibly with a degraded performance.

The problem we are considering here is the reconfiguration problem in the case of a fault in processing cores. Reconfiguration in this particular case implies remapping of tasks onto the remaining healthy cores. Upon receiving a fault signal via the fault detection mechanisms, an algorithm will decide on the new task assignment configuration. This has to be a very fast algorithm, in order not to disrupt operation for long. The study of migration cost in the context of real-time systems is not the focus of this paper. We are currently more focused on minimizing performance degradation rather than minimizing migration costs. The task remapping algorithm can work in two ways: *limited task migration* where only the tasks on the faulty core are migrated to other cores; and *unlimited task migration* where any task can be migrated. The former will have a shorter reconfiguration time due to less number of tasks being migrated. However it will most likely result in a more degraded performance. The latter will certainly require a long reconfiguration time while having higher chance of less/non- degraded performance.

We propose an optimal solution to online task remapping problem based on our ILP formulation for both limited and unlimited task migration cases, and five different heuristics for the limited task migration case. Obviously the ILP solution cannot be applied at run-time. However, it makes it possible to measure the quality of the heuristic methods.

4.1 Optimal Task Remapping

In the case of unlimited task migration, we are able to obtain the Pareto curves for all single fault scenarios by adding the *faulty core* constraint below to the original ILP formulation.

Constraint *(faulty core)*: Given a faulty node n_f, a new constraint is added to the ILP formulation that forbids mapping of tasks on the faulty node n_f.

$$\sum_{j=1}^{|V_t|} X_{fj}^{NT} = 0 \tag{8}$$

In the case of limited task migration, the below constraint should be added as well.

Constraint *(migrate only tasks on the faulty core)*: Given a faulty node n_f and an initial task mapping M^{NT}, a new constraint is added to limit the reconfiguration just to the tasks that are running on faulty node n_f.

$$X_{ij}^{NT} = M_{ij}^{NT},\ 1 \leq i \leq |V_a|,\ 1 \leq j \leq |V_t|,\ i \neq f \tag{9}$$

For an heterogeneous NoC with all IP cores being different from each other, there can be $|V_a|$ successive single faults, eventually leading to no remaining non-faulty cores. The total number of different configurations N, from all healthy cores to one healthy core is

$$N = \sum_{i=1}^{|V_a|} \binom{|V_a|}{i} - 1 = 2^{|V_a|} - 1$$

It means that we will need to calculate N Pareto curves for all these different scenarios. It is a heavy task even if it is done offline. One way of implementing the task remapping algorithm is by means of a look-up table where we keep the resulting optimal mappings for all Pareto curves of the N different configurations. Assuming we have p points in a Pareto curve, encoding such information would cost B bits calculated as

$$B = (2^{|V_a|} - 1)\, p\, |V_t|\, \lceil log(|V_a|) \rceil$$

For a case with $|V_a| = 9$, $|V_t| = 12$, $p = 5$, we have $B = 14.97$ Kbytes. As the number of cores with same type increases and also depending on their placements in the NoC, this number decreases due to the occurrences of symmetrical configurations. In case the local memory in the

tiles is restricted and/or the size of the NoC and the problem increase, this memory requirement may be prohibitive to apply the look-up table technique. However, it has been argued that such an offline technique can provide more predictable, faster reconfiguration times, and optimal degradations [13].

4.2 Center of Gravity method (CoG)

This heuristic places the task to be migrated in a core that resides in between the other tasks it communicates with considering the amount of communication. This heuristic takes into account only communication cost. More formally, let $t_i \in L_f$ be the tasks that reside on the faulty core n_f. Let *peers* be a function that returns the list of tasks that a given task communicates with. Let *weight* be a function that returns the bandwidth demand between two given tasks. Let *coord* be a function that returns the (x,y) coordinates of a given node in mesh-based NoC. Let *map* be a function that returns the node of the given task. We transform the problem to finding the center of gravity of masses by considering the weights of communication as the masses of the peer tasks. Then the new node n_i for task $t_i \in L_f$ will have coordinates $coord(n_i)$

$$coord_i = \frac{\sum_{t_j \in peers(t_i)} coord(\, map(t_j)\,)\; weight(t_j, t_i)}{\sum_{t_j \in peers(t_i)} weight(t_j, t_i)}, t_i \in L_f$$

It is most likely that $coord_i$ will not have integer values, so we round it to obtain actual coordinates.

$$coord(n_i) = \lfloor coord_i + (0.5, 0.5) \rfloor$$

If n_i is not a core type that can realize the task t_i or it is a faulty core itself, then we look at the node in the close vicinity with minimum computational load. If there are tasks that communicate with each other and reside in the faulty node, the resulting mapping will depend on which order those tasks are being migrated. A logical decision is to sort the tasks with respect to their total bandwidth demands on their edges connected to the tasks on the non-faulty nodes and then migrate them in descending order.

4.3 Nonidentical Multiprocessor Scheduling

The objective regarding computation is equivalent to the scheduling of independent tasks on nonidentical processors in order to minimize the makespan. We adopt three heuristics that has been proposed in [10] for this problem. They are slightly different from each other and it has been shown that there are examples in which each of them is superior to others. However they have different orders of complexity, $O(n)$, $O(n \log n)$, $O(n^2)$. These heuristics consider only computation cost when remapping the tasks. It may be the case that the resulting remapping does not satisfy the capacity constraints (Equation 4).

NMS-A: L_f is the set of tasks to be migrated from the faulty node n_f. T_j^N is the sum of the execution times of tasks assigned to node n_j. Let L_j be the set of tasks assigned to core n_j. The execution time $T^{TN}_{cap\ ij}$ of task t_i if assigned to node n_j can be calculated in matrix form as

$$T^{TN}_{cap} = T^{TC}_{cap} M^{CN}$$

The task $t_i \in L_f$ is scheduled on the core that minimizes its finishing time. Inputs to the NMS-A algorithm is the initial mapping L and T^N before the fault occurrence. The output is the new mapping L.

Algorithm 1 NMS-A Algorithm

1: **for all** $t_i \in L_f$ **do**
2: find the smallest j such that $T_j^N + T^{TN}_{cap\ ij} \leq T_l^N + T^{TN}_{cap\ il}$ for all $1 \leq l \leq |V_a|$, $l \neq f$
3: $L_j \leftarrow L_j \cup \{t_i\}$, $L_f \leftarrow L_f \setminus \{t_i\}$
4: $T_j^N \leftarrow T_j^N + T^{TN}_{cap\ ij}$
5: **end for**
6: **return**

NMS-B: For each task $t_i \in L_f$, Algorithm NMS-B first orders the tasks in L_f according to decreasing $min\{T^{TN}_{cap\ ij} : 1 \leq j \leq |V_a|\}$, and then calls Algorithm NMS-A.

NMS-C: This algorithm iteratively schedules the tasks by choosing a task from L_f that gives the least finishing time.

Algorithm 2 NMS-C Algorithm

1: **while** $L_f \neq \emptyset$ **do**
2: find a task $t_i \in L_f$ and $n_j \in |V_a| \wedge n_j \neq n_f$ such that $T_j^N + T^{TN}_{cap\ ij} \leq T_j^N + T^{TN}_{cap\ kj}$ for all $t_k \in L_f$ and $T_j^N + T^{TN}_{cap\ ij}$ is minimum.
3: $L_j \leftarrow L_j \cup \{t_i\}$, $L_f \leftarrow L_f \setminus \{t_i\}$
4: **end while**
5: **return**

4.4 Localized NMS Heuristic (LNMS)

The heuristics proposed above take into account either communication or computation. In order to propose a heuristic that performs well for both objectives, we limit the region of nodes where we employ the NMS heuristics. This algorithm is called the Localized NMS (LNMS) where communication cost is bounded by selecting a remapping region for each task that falls in between the peer tasks. We define a *region* function that returns a list of nodes for a given task. All three NMS heuristics can have localized versions such as LNMS-A, LNMS-B and LNMS-C. Rather than rewriting the whole pseudocode, we highlight the differences in each case:

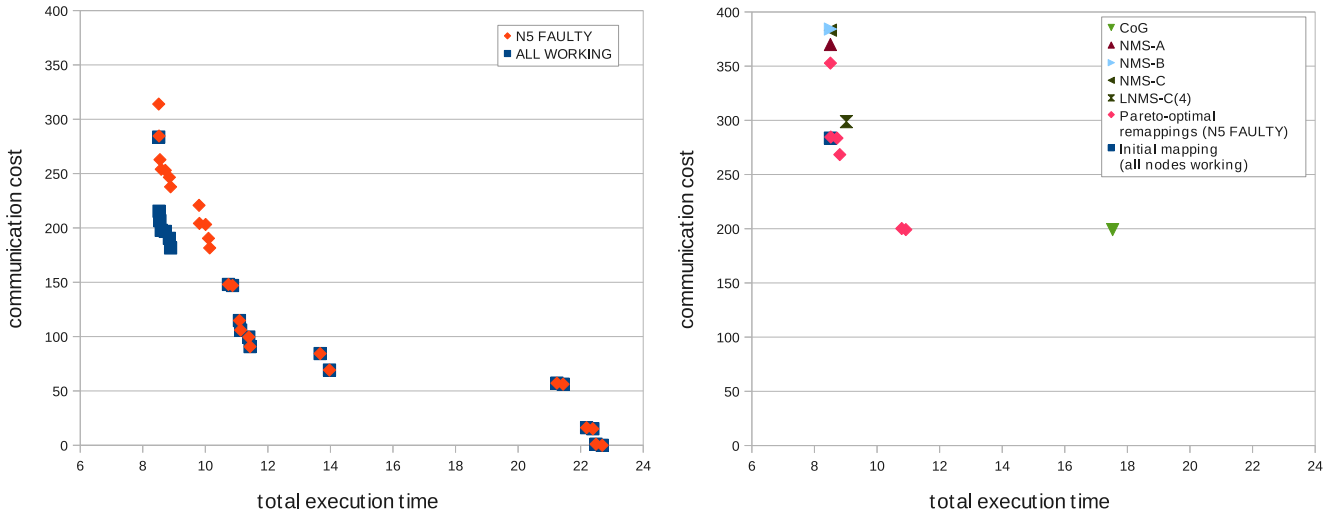
LNMS-A and LNMS-B: In line 2, instead of $1 \leq j \leq |V_a|$, we have $n_j \in$ region(t_i).

LNMS-C: In line 2, instead of $n_j \in V_a$, we have $n_j \in$ region(t_i).

We define a parametrized $region(t_i, s)$ function that takes an integer s and returns a set of s nodes making up a region centered around the center of gravity of the peer tasks of the given task t_i. For $s = 1$, LNMS reduces to CoG ($LNMS(1) \equiv CoG$) and for $s = |V_a|$, it reduces to NMS ($LNMS(|V_a|) \equiv NMS$). Therefore we should be able to obtain a sub-optimal Pareto curve for the range $1 \leq s \leq |V_a|$ that represents different trade-off points between communication and computation.

4.5 Results

For the case study described in section 3, we obtained Pareto curves for 3 different fault scenarios: one with faulty core n_1, one with faulty n_2 and one with faulty n_5. There are only 3 unique single fault scenarios for the 3x3 NoC in Figure 2 because of symmetry relations (faulty $n_1 \equiv$ faulty $n_3 \equiv$ faulty $n_7 \equiv$ faulty n_9 etc). Figure 4a shows the Pareto-optimal remappings in the case of faulty n_5 and unlimited task migration along with the Pareto-optimal mappings



(a) Pareto curves for optimal mappings and optimal remappings (unlimited task migration)

(b) Comparison of results between heuristics, optimal remappings and the initial mapping (limited task migration)

Figure 4: Remapping results for the MPEG2 decoder case study on a 3x3 mesh NoC-based heterogeneous multi-processor architecture with a faulty node n_5

when all nodes are working. The performance degradation is clearly visible especially for more constrained values of total execution time. This is simply because the required parallelism, thus number of nodes, increases for such values and n_5 plays a key role as the central node of the mesh.

The limited task migration case has also been considered. Starting with an initial optimal mapping ($t_7, t_8, t_9, t_{10} \rightarrow n_1; t_{11} \rightarrow n_2; t_3, t_4, t_5, t_6 \rightarrow n_3; t_1, t_2, t_{12} \rightarrow n_5$), the Pareto-optimal remappings have been calculated for 3 different cases: faulty n_1, faulty n_2 and faulty n_5. Table 3 lists Pareto-optimal remappings in limited case for the 3 fault scenarios along with the performance degradation ratios with respect to the initial mapping situation prior to the fault.

We have also calculated the remappings for the three fault scenarios by using the CoG, NMS-A, NMS-B, NMS-C and LNMS-C(4) heuristics. Table 4 lists these results providing also the degradation with respect to the initial mapping. The results for limited task remapping in the fault scenario of n_5 is visualized in Figure 4b. It shows the proximity of the remapping results of 5 heuristics to the metrics of the initial mapping and the Pareto-optimal remappings. The results reveal that LNMS-C(4) gives a remapping that is the closest to the optimal remapping values for all fault scenarios incurring at most 6% more degradation than Pareto-optimal remappings for both of the objectives.

Table 3: Degradation achieved by Pareto-optimal limited remappings for all single fault scenarios

faulty node	mapping	exe.time (obj. 1)	com.cost (obj. 2)	degradation	
				obj. 1	obj. 2
none	initial	8.51	283.6	-	-
n_1	Pareto1	8.51	305.2	1.00	1.08
	Pareto2	8.55	296.4	1.00	1.05
	Pareto3	9.05	261.8	1.06	0.92
	Pareto4	9.80	205.6	1.15	0.73
	Pareto5	11.09	184.0	1.30	0.65
n_2	Pareto1	8.51	413.6	1.00	1.46
	Pareto2	16.44	298.8	1.93	1.05
n_5	Pareto1	8.51	352.8	1.00	1.24
	Pareto2	8.52	284.6	1.00	1.00
	Pareto3	8.72	283.6	1.02	1.00
	Pareto4	8.81	268.4	1.04	0.95
	Pareto5	10.79	200.2	1.27	0.71
	Pareto6	10.92	199.2	1.28	0.70

5. CONCLUSION AND FUTURE WORK

We formulated the optimal task mapping problem for mesh-based NoC multiprocessors with deterministic routing as an integer linear programming (ILP) problem with the objective of minimizing the communication traffic in the system and the total execution time of the application. We used it to obtain optimal task remappings in presence of faults in processing cores. Several heuristics has been proposed and compared with respect to the optimal remappings. We have found out that LNMS-C has performed the best (within 6% proximity to optimum) in the MPEG2 decoder case study with three fault scenarios. We wish to continue this work by evaluating our heuristics on more applications, larger NoC dimensions and more fault scenarios. Although our examples involved single fault scenarios, the proposed technique can be applied in presence of successive faults.

Acknowledgment

This work was funded by the European Commission under the Project MADNESS (No. FP7-ICT-2009-4-248424). The authors would like to thank Mariagiovanna Sami and Z. Caner Taskin for their valuable comments on this work.

Table 4: Degradation achieved by proposed heuristics for all single fault scenarios

faulty node	Approach	exe.time (obj. 1)	com.cost (obj. 2)	degradation	
				obj. 1	obj. 2
none	initial	8.51	283.6	-	-
n_1	CoG	10.99	296.4	1.29	1.05
	NMS-A	8.51	603.8	1.00	2.13
	NMS-B	8.51	603.8	1.00	2.13
	NMS-C	8.51	482.6	1.00	1.70
	LNMS-C(4)	8.51	314	1.00	1.11
n_2	CoG	16.44	298.8	1.93	1.05
	NMS-A	8.51	413.6	1.00	1.46
	NMS-B	8.51	413.6	1.00	1.46
	NMS-C	8.51	413.6	1.00	1.46
	LNMS-C(4)	8.51	413.6	1.00	1.46
n_5	CoG	17.53	199.2	2.06	0.70
	NMS-A	8.51	370.0	1.00	1.30
	NMS-B	8.51	384.2	1.00	1.35
	NMS-C	8.51	383.2	1.00	1.35
	LNMS-C(4)	9.02	298.8	1.06	1.05

6. REFERENCES

[1] C. Ababei and R. Katti. Achieving network on chip fault tolerance by adaptive remapping. *Int. Parallel and Distributed Processing Symposium*, 0:1–4, 2009.

[2] G. Ascia, V. Catania, and M. Palesi. Multi-objective mapping for mesh-based noc architectures. In *Int. Conf. on Hardware/Software Codesign and System Synthesis*, pages 182 – 187, Sep 2004.

[3] K. Bhardwaj and R. Jena. Energy and bandwidth aware mapping of ips onto regular noc architectures using multi-objective genetic algorithms. In *Int. Sym. on System-on-Chip*, pages 27–31, Oct 2009.

[4] V. Chankong and Y. Haimes. *Multiobjective Decision Making Theory and Methodology*. North-Holland, 1983.

[5] C.-L. Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. In *IEEE Int. Conf. on Computer Design*, pages 164–169, 2008.

[6] C. Erbas, S. Cerav-Erbas, and A. Pimentel. Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. *IEEE Tran. on Evolutionary Computation*, 10(3):358–374, June 2006.

[7] A. R. Fekr, A. Khademzadeh, M. Janidarmian, and V. S. Bokharaei. Bandwidth/fault tolerance/contention aware application-specific noc using pso as a mapping generator. In *Proc. of The World Congress on Engineering*, pages 247–252, 2010.

[8] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Proc. of the Asia and South Pacific Design Automation Conf.*, pages 233 – 239, Jan 2003.

[9] J. Hu and R. Marculescu. Energy and performance aware mapping for regular noc architectures. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562, April 2005.

[10] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. ACM*, 24:280–289, April 1977.

[11] R. K. Jena and P. K. Mahanti. Design space exploration of network-on-chip - a system level approach. *Int. J. of Computing and ICT Research*, 2:17–25, June 2008.

[12] S. Le Beux, G. Bois, G. Nicolescu, Y. Bouchebaba, M. Langevin, and P. Paulin. Combining mapping and partitioning exploration for noc-based embedded systems. *J. Syst. Archit.*, 56:223–232, 2010.

[13] C. Lee, H. Kim, H.-w. Park, S. Kim, H. Oh, and S. Ha. A task remapping technique for reliable multi-core embedded systems. In *Proc. of the Eighth Int. Conf. on Hardware/software codesign and system synthesis*, pages 307–316, 2010.

[14] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proc. of Euromicro Symposium on Digital System Design*, pages 180 – 187, Sep 2003.

[15] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, and F. Hessel. Exploring noc mapping strategies: an energy and timing aware technique. In *Proc. of DATE*, pages 502–507, March 2005.

[16] R. Marculescu. Networks-On-Chip: The Quest for On-Chip Fault-Tolerant Communication. In *Proc. of ISVLSI*, page 8, Washington, DC, USA, 2003.

[17] M. Modarressi and H. Sarbazi-Azad. Power-aware mapping for reconfigurable noc architectures. In *25th Int. Conf. on Computer Design*, pages 417 –422, 2007.

[18] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. In *Proc. of the Design Automation and Test Europe Conf.*, volume 2, pages 896–901, Feb 2004.

[19] K. Srinivasan and K. Chatha. A technique for low energy mapping and routing in network-on-chip architectures. In *Proc. of the Int. Symposium on Low Power Electronics and Design*, pages 387 – 392, 2005.

[20] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear-programming-based techniques for synthesis of network-on-chip architectures. *IEEE Trans. Very Large Scale Integr. Syst.*, 14:407–420, April 2006.

[21] L. Thiele, I. Bacivarov, W. Haid, and K. Huang. Mapping applications to tiled multiprocessor embedded systems. In *Seventh Int. Conf. on Application of Concurrency to System Design*, pages 29 –40, July 2007.

[22] I. Walter, I. Cidon, A. Kolodny, and D. Sigalov. The era of many-modules soc: revisiting the noc mapping problem. In *2nd Int. Workshop on Network on Chip Architectures*, pages 43 –48, Dec 2009.

[23] X. Wang, M. Yang, Y. Jiang, and P. Liu. A power-aware mapping approach to map ip cores onto nocs under bandwidth and latency constraints. *ACM Trans. Archit. Code Optim.*, 7:1–30, 2010.

[24] Y. Yi, W. Han, X. Zhao, A. T. Erdogan, and T. Arslan. An ilp formulation for task mapping and scheduling on multi-core architectures. In *Proc. of the Design Automation and Test Europe Conf.*, pages 33–38, 2009.

[25] W. Zhou, Y. Zhang, and Z. Mao. Pareto based multi-objective mapping ip cores onto noc architectures. In *IEEE Asia Pacific Conf. on Circuits and Systems*, pages 331 –334, Dec 2006.