

# Communication and Task Scheduling of Application-Specific Networks-on-Chip\*

Jingcao Hu

Radu Marculescu

Carnegie Mellon University

Pittsburgh, PA 15213-3890, USA

email: {jingcao, radum}@ece.cmu.edu

## Abstract

The objective of this paper is to introduce a novel Energy-Aware Scheduling (EAS) algorithm which statically schedules application-specific communication transactions and computation tasks onto heterogeneous Network-on-Chip (NoC) architectures. The proposed algorithm automatically assigns the application tasks onto different processing elements and then schedules their execution under real-time constraints. At the same time, the algorithm takes into consideration the exact communication delay by scheduling communication transactions in parallel. As the main theoretical contribution, we first formulate the problem of concurrent communication and task scheduling for heterogeneous NoC architectures and then propose an efficient heuristic to solve it. Experimental results show that significant energy savings can be achieved while meeting the specified performance constraints. For instance, for a complex multimedia application, 31% energy savings have been observed, on average, compared to the schedules generated by a standard earliest-deadline-first scheduler.

## 1 Introduction

Regular NoC architectures have been recently proposed as a promising solution to the increasingly complex on-chip communication problems [1][2]. Such chips implementing the NoC approach consist of an array of regular tiles where each tile can be a general-purpose processor, a DSP, a memory subsystem, *etc* (e.g. [4][5][7]). A router is typically embedded within each tile and thus, instead of routing design-specific global on-chip wires, the inter-tile communication can be achieved by routing packets. With the increase of IP integration and component specialization, the heterogeneity of these designs inevitably increases; this heterogeneity helps not only in optimizing the overall performance for low-power consumption, but also in ensuring competitive design costs.

---

\*Research supported by NSF CCR-00-93104 and DARPA/Marco Gigascale Research Center (GSRC)

Although sharing some similarities (e.g. topology, packetized routing, *etc.*), with traditional direct networks in multicomputers [25] (referred to as *macro-networks* hereafter), the NoC-based communication paradigm has specific properties which differentiate it from their macro-network counterparts. More specifically:

- Compared to typical macro-networks, an on-chip network is by far more *resource limited*. To minimize the implementation costs, the on-chip network should be implemented with very little area overhead. This is especially important for those architectures composed of tiles designed at a fine-level of granularity. As we will see later in the paper, this has a significant impact on the architectural design considerations.
- Unlike most of the macro-network designs in which the only goal is to achieve the highest possible performance (in terms of throughput, average packet latency, *etc.*), most NoC designs are constrained by the *power consumption* budget. As the market of mobile products continues expanding and the chip cooling costs keep increasing, low power design becomes increasingly important. Indeed, for a large class of products (e.g. portable multimedia devices, mobile phones), power consumption is the primary optimization objective, while the performance is typically specified as a design constraint.
- In contrast to typical macro-networks which provide general platforms for a large spectrum of applications, most NoCs are developed *specifically* for one application or as a platform for a small class of applications. Consequently, the designer has a good understanding of the traffic characteristics and can use this information to customize the NoC design accordingly.

Because of the aforementioned differences, the existing techniques for macro-networks optimization can not be directly applied to the NoC context. Thus, there is a need for a NoC-specific design methodology which is critical to fully exploit the advantages the NoC paradigm has to offer. As a consequence, research in this direction has emerged in both circuit design and design automation research communities. Based on the potential for customization, the NoC platforms can be roughly classified into three classes: *hard*, *firm* and *soft* (see Fig. 1).

*Hard NoC* platforms correspond to having totally fixed architectures where both computation and communication components have been pre-designed; thus, they offer no real flexibility for architectural customization and the customization process reduces essentially to solving the communication and task scheduling problem.

*Firm NoCs* correspond to platforms where the communication architecture has been pre-designed but the designer still has the choice of deciding how to embed different IPs onto different tiles which act as placeholders in the architecture. The new problem that needs to be solved for customizing the firm platforms is the *mapping and routing path allocation*; that is, one needs to decide how to map different IPs onto different tiles and how to allocate routing paths such that the metrics of interest (e.g. energy) are minimized.

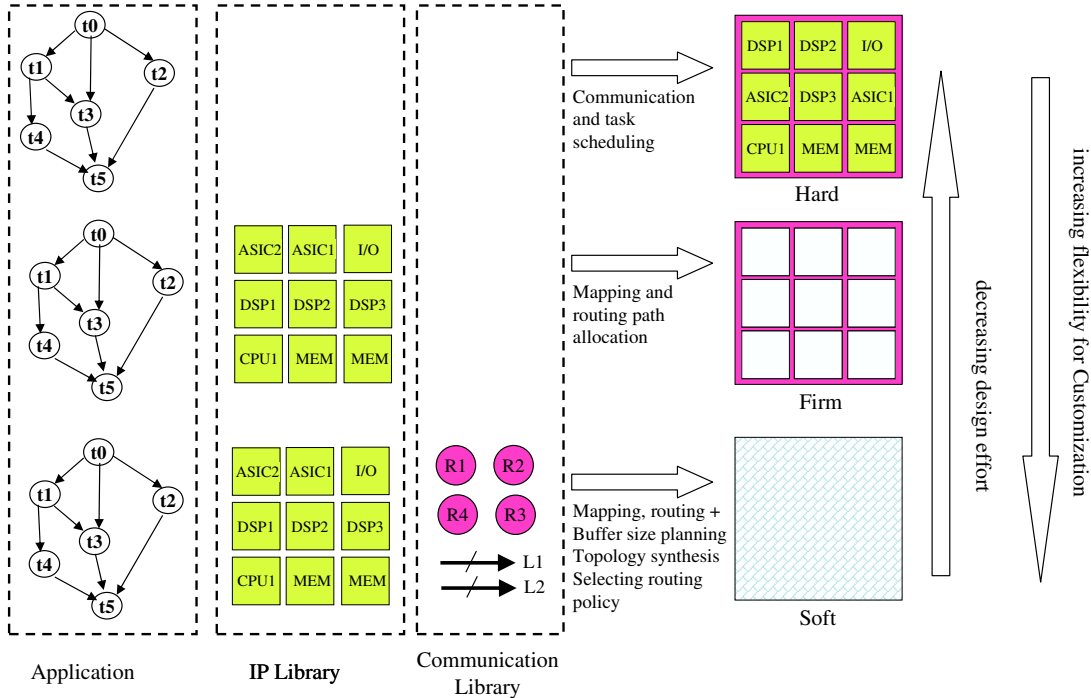


Figure 1: The customization problem for NoCs

Finally, *soft NoC* platforms offer the maximum flexibility for customization<sup>1</sup>. The designer has the freedom of choosing the communication components from a given library and later connect them with other IPs to construct the entire NoC. The soft NoC platforms offer designers the most flexibility in matching the architecture with the characteristics of the application. The trade off is that the platforms can no longer be pre-optimized as the hard or firm platforms since some design factors (e.g. floorplan, wire length, *etc.*) can not be pre-determined with reasonable accuracy. The customization process for such platforms includes selecting the appropriate routers from the library, customizing the network (e.g. choose the network topology, determine the router buffer sizes), *etc*, for optimizing a particular metric of interest.

In this paper, we concentrate on the first category of NoC platforms (that is, *hard* NoCs) and present a novel scheduling algorithm for such systems. In particular, we focus on embedded systems running on batteries where maximizing the battery-life is the main design objective. On the other hand, another essential characteristic is their timing behavior; this is usually specified by *deadlines* associated to individual tasks.

For performance evaluation purposes, any application running on an embedded system can be described as a *Communication Task Graph* (CTG) (see the right hand side of Fig. 2 for a simplified representation). The tasks in the CTG can have two types of dependencies:

<sup>1</sup>Strictly speaking, this is not a platform since even the communication architecture is yet to be decided based on the selected communication components.

*control dependencies* and *data dependencies*. The control dependencies indicate that one task can not start its execution until another task has finished, while the presence of data dependencies implies that these tasks communicate with each other.

Given a CTG and a pre-selected architecture, one important problem is how to schedule the computation tasks and the communication transactions onto the target architecture. This includes *i*) deciding on the *assignment* of tasks and communication transactions onto different computation and communication resources, respectively, and *ii*) fixing the *order* of their execution on these shared resources. We refer to this as the “scheduling problem” for NoC architectures. The solution to the scheduling problem has a significant impact on the total system energy consumption because:

- Due to the heterogeneity of the architecture, assigning the same task to different processing elements (PEs) (e.g. PowerPC vs. DSP cores) leads to very different energy consumption values associated to computation.
- For different task assignments, the inter-task communication volume and the routing path can vary significantly; this leads to very different values for the communication energy consumption.

Although the scheduling problem is a traditional research topic, almost all previous work focuses on maximizing the performance through the scheduling process. The algorithms developed this way are thus *not* suitable for real-time embedded applications where the objective is to minimize the energy consumption under tight performance constraints. Moreover, most previous work neglects the inter-processor communication aspects during the scheduling process, or just assumes a fixed delay proportional to the communication volume, without taking into consideration subtle effects like communication congestion which may change dynamically throughout tasks execution. As we will show later in the paper, considering communication effects turns out to be critical for NoC architectures.

In this paper, we propose a new algorithm for scheduling both *communication and computation* in heterogeneous NoC architectures with the primary objective of minimizing the system energy consumption. At the same time, the algorithm handles arbitrary communication and execution costs for the application, schedules tasks and communication transactions by considering the network congestion, as well as PE’s heterogeneity. The novelty of our work can be summarized as follows:

- Instead of using only performance as the optimization objective, our algorithm tries to minimize the energy consumption of the application under tight performance constraints.
- The communication scheduling *and* the computation task scheduling are carried out in parallel; therefore, the obtained scheduling results can be very accurate because they take into consideration the effects of the traffic dynamics. This is critical for real-time applications.

- The target architecture is heterogeneous; that is, the generic architecture we consider is composed of PEs with different functionalities, power and performance figures.

The remainder part of this paper is organized as follows. We first review the related work (Sec. 2) and present a concise description of the NoC architecture and its energy model (Sec. 3). The problem of energy-aware communication and task scheduling is then formulated in Sec. 4. Next, an efficient heuristic algorithm based on slack budgeting is proposed to solve this problem under performance constraints (Sec. 5). In Sec. 6, we report our experimental results using random, as well as real multimedia, task graphs. Finally, we conclude by summarizing our main contribution.

## 2 Related Work

NoC-based communication has recently attracted significant attention because of its potential for performance improvement, reuse capability and power efficiency. As a result, the regular NoC architectures have been discussed extensively [1][4][5], NoC platforms have been prototyped [7][10][11] and simulation tools/models have been already developed [12].

In terms of the customization techniques for NoCs, an efficient mapping algorithm is proposed in [17] to map IPs onto a tile-based NoC architecture such that the total communication energy consumption is minimized. At the same time, the performance of the mapped system is guaranteed to satisfy the specified constraints through bandwidth reservation. The approach is further improved in [18] by taking into consideration the packet routing flexibility during the mapping process. From a different angle, in [19], a fast algorithm is developed which also maps IP onto a mesh NoC architecture but with the different objective of minimizing the average communication latency. However, all these customization techniques focus on *firm* NoCs, while the work presented in the paper is devoted to *hard* NoCs by proposing an efficient scheduling algorithm.

In the area of scheduling research, Eles *et al* in [13] present a scheduling algorithm for distributed embedded systems which takes into consideration bus access optimization. Sih *et al* [14] propose a compile-time scheduling heuristic called *dynamic-level scheduling* which accounts for interprocessor communication overhead. Both papers try to maximize the performance of the system without considering the system energy consumption.

From another perspective, most of the work on low-power scheduling (e.g. [6][15]) focuses on architectures with dynamic voltage scaling or dynamic power management capabilities and tries to exploit the task execution slacks. The authors assume homogeneous, shared-bus architectures and thus, their work can not be applied in our scenario where the target architecture is totally different.

## 3 Platform Characterization

Although our algorithm can easily be applied to architectures with different network topologies, in this paper, we focus on the tile-based NoC architecture as an illustrative platform for our work. In this section, we describe the suitable routing schemes for NoCs, the regular tile-based architecture and the energy model for its communication network.

### 3.1 Routing Issues

In order to be able to direct the information appropriately, a tile-based architecture requires a method of routing the data packets through the network. In the following, we adapt macro-network routing techniques to the context of NoCs by taking into consideration the major differences between macro-networks and NoCs.

#### 3.1.1 Buffering space

To minimize the implementation cost, the on-chip network has to be implemented with little area overhead. This is especially critical for those architectures composed of tiles defined at a fine-level of granularity. Thus, instead of using huge memories (e.g. SRAM or DRAM) as buffering space for the routers/switches as is the case in macro-networks, it's more reasonable to use register-based buffers for on-chip routers. Another advantage of using registers over huge memories is that the access latency can be significantly reduced. This is critical for latency sensitive applications which are typical in many SoCs.

#### 3.1.2 Wormhole routing

Because of the limited buffering resources that are available and the stringent latency requirements that characterize typical NoC applications, we believe that wormhole-based routing [3] is the most appropriate routing technique for NoCs. In wormhole routing, a packet is divided into *flow control digits (flits)* as basic units. The flits are then routed through the network in a pipelined fashion, which leads to a dramatically reduced communication latency. The first flit in a packet (the header flit) contains all the routing information and leads the packet through the network. When the header flit is blocked due to the congestion somewhere in the network, all of the trailing flits will wait at their current nodes. Thus, large packet buffers at each intermediate node are obviated and only a small buffer of flit size is required.

#### 3.1.3 Deterministic routing

There are quite a few wormhole routing techniques published to date. In general, they can be classified into two categories: *deterministic* and *adaptive* routing [8]. We believe that for NoC, deterministic routing is more suitable because of:

- *Resource limitation and stringent latency requirements*

Compared to deterministic routers, implementing adaptive routers requires by far more resources. Moreover, since in adaptive routing packets may arrive out of order, huge buffering space is needed to reorder them. This, together with its protocol overhead, leads to prohibitive costs, extra delay and jitter.

- *Traffic predictability*

In contrast to typical direct macro-networks which provide *general* platforms for a large spectrum of applications, most NoCs are developed just for one application or, at most, a small class of applications. Consequently, the designer may have a good understanding of the traffic characteristics and can apply this information to avoid congestion by wisely mapping the IPs and allocating the routing paths.

### 3.1.4 Freedom of deadlock and livelock

A desirable feature of a routing algorithm is its freedom from *deadlock* and *livelock* [24]. All deterministic and minimal<sup>2</sup> routing algorithms are livelock-free. Freedom from deadlock is especially critical for NoCs. Indeed, implementing a mechanism which automatically detects and recovers from deadlock may be not only too expensive in terms of silicon resources, but also it may lead to unpredictable delays.

In summary, we argue that the appropriate routing technique for NoCs should be *deterministic, deadlock-free, minimal, wormhole-based*.

## 3.2 Architectural issues

The system under consideration is composed of  $n \times n$  tiles interconnected by a  $2D$  mesh network. The left part of Fig. 2 shows the abstract view of the NoC architecture consisting of 16 tiles ( $p_0, p_1, \dots, p_{15}$ ) which has been used recently by several authors [1][5].

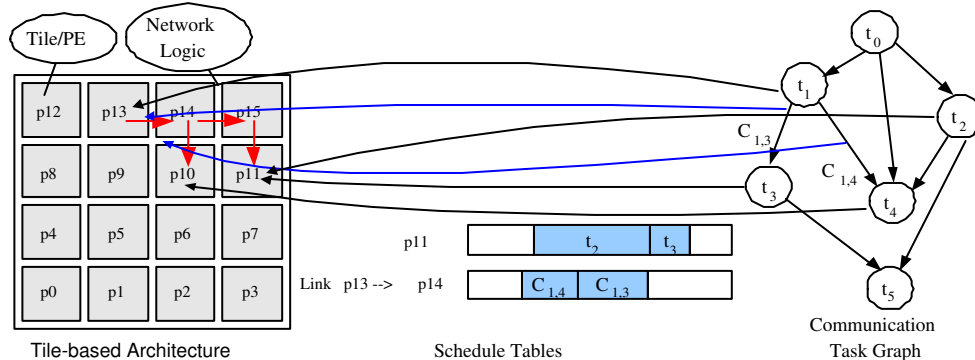


Figure 2: A generic tile-based NoC architecture and its scheduling problem

<sup>2</sup>Minimal refers to the routing of packets only along the shortest paths.

Each tile in Fig. 2 is composed of a *processing element* (PE) and a *router* (Fig. 3). The PEs embedded in the tiles of the NoC are assumed to be heterogeneous. For instance, one tile can be a DSP, another tile can be a high performance, energy-hungry CPU, yet another one can be a low-power ARM processor. Because of the different functionality and performance characteristics, it takes different amounts of computation time and computation energy to execute the very same task on different tiles.

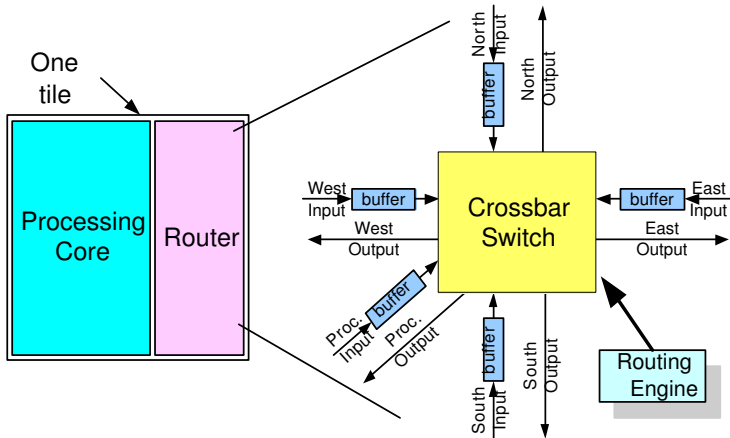


Figure 3: The typical structure of a tile

Due to the limited resources, the buffers in the routers are implemented using registers and can typically hold one or several flits each. Similar to [5][7], a  $5 \times 5$  crossbar switch is used as the switching fabric in the router. For the sake of simplicity, the XY routing scheme is used to direct the packets across the chip. In short, for 2D mesh networks, XY routing first routes packets along the X-axis. Once it reaches the column wherein lies the destination tile, the packet is routed along the Y-axis. Obviously, the XY routing is a *minimal* path routing algorithm and is *free* of deadlock and livelock [24].

Once a schedule has been determined, the architecture must be compiled such that all the PEs and routers will coordinate together under the given schedule. In order to achieve this, we assume that the PEs have the capability to control at what time to start the execution of a given task and at what time to start sending a message. Such capability is usually already provided by the OS hosted on the PE and is a common assumption in scheduling research (e.g. [13][14]).

### 3.3 Energy modeling issues

Energy modeling for NoC architectures was a relatively unexplored area until recently. Ye *et al* in [16] proposed a energy model for network routers by defining the *bit energy* ( $E_{bit}$ ) metric as the energy consumed when one bit of data is transferred through the router. For NoCs with buffers implemented by registers, both Hu *et al* in [17] and Ye *et al* in [20]

suggest calculating  $E_{bit}$  as:

$$E_{bit} = E_{R_{bit}} + E_{L_{bit}} \quad (1)$$

where  $E_{R_{bit}}$  and  $E_{L_{bit}}$  represent the energy consumed on the router and on the link between PEs, respectively.

Thus, by using Eq. (1), the average energy consumption for sending one bit of data from PE  $p_i$  to PE  $p_j$  can be analytically calculated as:

$$E_{bit}^{p_i, p_j} = n_{hops} \times E_{R_{bit}} + (n_{hops} - 1) \times E_{L_{bit}} \quad (2)$$

where  $n_{hops}$  is the number of routers the bit traverses on its way from PE  $p_i$  to PE  $p_j$ .

For  $2D$  mesh networks with *minimal* routing, Eq. (2) shows that the average energy consumption of sending one bit of data from PE  $p_i$  to PE  $p_j$  is determined by the *Manhattan* distance between them. Although more accurate energy models exist in the literature (e.g. [21]), in this paper, we choose the energy model described in Eq. (1) as it provides an efficient approximation for the NoC architectures under consideration with reasonable accuracy at this level of abstraction.

## 4 Problem Formulation

Simply stated, given the CTG of an application and a target NoC architecture, the problem we need to solve is to find a *feasible, non-preemptive, static* schedule such that the application energy consumption is minimized under tight performance constraints. More specifically, this includes:

1. Determining *which PE* in the NoC architecture should each task  $t_i$  be scheduled to, and *the particular time slot* when the task will be executed on that PE. For example in Fig. 2, this means to determine to which PE (e.g.  $p_{10}$ ,  $p_{11}$  etc.) each task (e.g.  $t_2$ ,  $t_3$ ) should be assigned to. If  $t_2$  and  $t_3$  are both assigned to the same PE  $p_{11}$ , for instance, our approach should also determine which one should be executed first and when exactly it should be executed.
2. Determining the time slots for all the communication transactions in the application. For instance, if tasks  $t_1$ ,  $t_3$  and  $t_4$  are assigned to PEs  $p_{13}$ ,  $p_{10}$  and  $p_{11}$ , respectively, then we need to determine the execution order of communication transaction  $c_{1,3}$  and  $c_{1,4}$  on the link from PE  $p_{13}$  to PE  $p_{14}$ , assuming that XY routing is used.

To formulate this problem, we define the following terms:

**Definition 1:** A *Communication Task Graph* (CTG)  $\mathcal{G} = G(T, C)$  is a *directed acyclic* graph, where each vertex represents a computational module of the application referred to as a task  $t_i \in T$ . Each  $t_i$  has the following properties:

- An array  $R^i$ , where the  $j$ -th element  $r_j^i \in R^i$  gives the execution time of task  $t_i$  if it is executed on  $j$ -th PE in the architecture.

- An array  $E^i$ , where the  $j$ -th element  $e_j^i \in E^i$  gives the energy consumption of task  $t_i$  if it is executed on  $j$ -th PE in the architecture.
- A deadline  $d(t_i)$  which represents the time when  $t_i$  has to finish. If the designer does not specify a deadline for task  $t_i$ , then  $d(t_i)$  is taken equal to infinity.

Each directed arc  $c_{i,j} \in C$  characterizes the communication or control dependency between tasks  $t_i$  and  $t_j$ . The arrow pointing out from  $t_i$  to  $t_j$  indicates that the task  $t_j$  can not start before  $t_i$  is finished. Each  $c_{i,j}$  has associated with it  $v(c_{i,j})$ , which stands for the communication volume (*bits*) from  $c_i$  to  $c_j$ . A non-zero value of  $v(c_{i,j})$  denotes that  $t_j$  can start only after  $t_i$  has finished and transferred  $v(c_{i,j})$  bits of data to task  $t_j$ .

Note that such similar application models are widely adopted in the area of scheduling research. For instance, CTG is equivalent to the *acyclic precedence expansion graph* model used in [14], the *task precedence graph* model used in [26] and the *task graph* model used in [9].

Obviously, to effectively exploit the parallelism provided by the NoC, the application has to be partitioned at the right level of granularity. Partitioning the application has to provide enough parallelism so that all the PEs in the architecture can be used more efficiently. Moreover, the granularity also directly affects the storage space requirements in the PEs as the messages need to be buffered. Since this focus here is on communication and computation scheduling, we will not address the application partitioning and modeling issues. Instead, we assume that the application to be scheduled are already partitioned and modeled in the form of a CTG.

**Definition 2:** An *Architecture Characterization Graph* (ACG)  $\mathcal{G}' = G(P, R)$  is a *directed* graph, where each vertex  $p_i \in P$  represents one PE in the architecture, and each directed arc  $r_{i,j} \in R$  represents the route taken from PE  $p_i$  to  $p_j$ . Each  $r_{i,j}$  has associated with it two metrics,  $e(r_{i,j})$  and  $b(r_{i,j})$ .  $e(r_{i,j})$  stands for the average energy consumption (in joules) of sending one bit of data from PE  $p_i$  to  $p_j$ , *i. e.*,  $E_{bit}^{p_i, p_j}$  when the route  $r_{i,j}$  is taken.  $b(r_{i,j})$  gives the bandwidth (in bits/second) of that route.

**Definition 3:** Communication transaction  $c_{i_1, i_2}$  is said to be *compatible* with another communication transaction  $c_{j_1, j_2}$  if and only if their execution times do not overlap or their routing paths do not intersect.

**Definition 4:** Task  $t_i$  is said to be compatible with task  $t_j$  if and only if their execution times do not overlap or they are assigned onto different PEs.

Using these definitions, the energy-aware scheduling problem for heterogeneous NoC architectures under real-time constraints can be formulated as:

**Given** a CTG and an ACG, **find** a mapping function  $\mathcal{M}()$  from tasks ( $T$ ) to PEs ( $P$ ), together with a start time for each task and communication transactions which minimizes energy consumption:

$$\min\{Energy = \sum_{\forall t_i} e_{\mathcal{M}(t_i)}^i + \sum_{\forall c_{i,j}} v(c_{i,j}) \times e(r_{\mathcal{M}(t_i), \mathcal{M}(t_j)})\} \quad (3)$$

such that

- All tasks are compatible with each other.
- All communication transactions are compatible with each other.
- All the control/data dependencies are satisfied.
- All the tasks  $t_i$  for which deadlines  $d(t_i)$  are specified finish the execution before or at their specified deadlines.

Since finding an optimal schedule for a multi-processor system that consumes the minimum energy is known to be NP-hard [22], in the following, we propose a heuristic algorithm which is capable of finding satisfactory solutions with reasonable short computation time.

## 5 Energy-Aware Scheduling

The energy-aware scheduling (EAS) approach is based on the idea of *slack-budgeting* which allocates more slack to those tasks whose mapping onto PEs has a larger impact on energy consumption and performance of the application. More specifically, our proposed approach performs the following three steps during the scheduling:

### Step 1. Budget slack allocation for each task

1. For each task  $t_i$ , three metrics are calculated:  $VAR_{e_i}$  is the variance of the energy consumption of  $t_i$  on different PEs, and  $VAR_{r_i}$  is the variance of the execution time of  $t_i$  on different PEs. Finally,  $M_{t_i}$  is the mean execution time of task  $t_i$  on different PEs.
2. Using the calculated  $VAR_{e_i}$  and  $VAR_{r_i}$ , task  $t_i$  is assigned a weight  $W_{t_i} = VAR_{e_i} \times VAR_{r_i}$ . This weight plays an important part in our algorithm. Intuitively, the larger this weight, the higher the priority the task should have in selecting the PE to be scheduled onto (this is because the execution of that task has a higher impact on the energy consumption and the overall performance of the system).
3. Using the  $M_{t_i}$  values for different tasks, we can calculate the slack for different paths, and then allocate the slack to different tasks based on their respective weights.

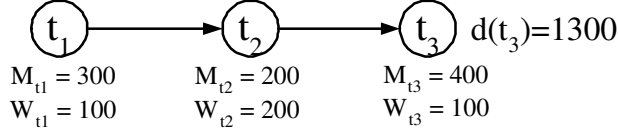


Figure 4: An example for budget slack allocation

For example, as shown in Fig. 4, tasks  $t_1$ ,  $t_2$  and  $t_3$  have the mean execution time of 300, 200 and 400 time units, respectively. Now, suppose that using the step 1 and 2 above, the weight of these tasks are calculated to be 100, 200 and 100, respectively. If  $t_3$  has a deadline of 1300 time units, then there will be a total slack of 400 time units for this path. The slack will be allocated to these three tasks proportionally to their weights. Thus, the slack for  $t_1$ ,  $t_2$  and  $t_3$  will be 100, 200 and 100 time units, respectively.

4. With these numbers, the *budgeted deadline* (BD) of each task is calculated. For example, in Fig. 4, the BD of task  $t_1$ ,  $t_2$  and  $t_3$  will be 400, 800 and 1300 respectively. We use  $BD_i$  to denote the budgeted deadline of task  $t_i$ .

## Step 2. Level based scheduling

1. Generate the *Ready Tasks List*; that is, list the tasks for which the precedent tasks have already been scheduled. In the following, this list will be referred to by RTL.
2. Let  $F(i, k)$  be the *earliest finish time* of task  $t_i$  if it is assigned to PE  $p_k$ . For each combination of task  $t_i \in RTL$  and PE  $p_k \in P$ , calculate its  $F(i, k)$  as:

$$F(i, k) = DRT(i, k) + r_k^i \quad (4)$$

where  $r_k^i$  is the execution time of task  $t_i$  when running on PE  $p_k$ .

In Eq. (4), DRT represents the *data ready time*, which is defined as the latest arrival time of all the receiving communication transactions of the corresponding task.

Obviously, to calculate  $DRT(i, k)$ , all the receiving communication transactions of  $t_i$  have to be scheduled. Given the list of the receiving communication transactions (LCT) of the task under consideration, the pseudo code shown in Fig. 5 is used to schedule the execution of these transactions.

In Fig. 5, the transactions in the LCT are first sorted by sender tasks' finish time. For each transaction, the path that it uses is then determined and the schedule table of that path is built by merging all the occupied slots of its comprising links. The transaction is finally scheduled to the earliest time, while satisfying the current schedule table of the path.

We should note that in this process, the communication transactions and tasks are only scheduled for the purpose of calculating  $F(i, k)$ . The schedule tables for both

```

sort LCT by the finish time of its sender;
for each trans in LCT {
    path = get_path(trans);
    dur = trans.bandwidth();
    path.build_schedule_table();
    sender_ft = trans.sender.finish_time();
    start_t=path.schedule_table.find_earliest(sender_ft,dur);
    for each link in path
        link.update_schedule_table(start_t,dur);
}

```

Figure 5: The communication scheduling strategy

links and the PEs will be restored every time when a new  $F(i, k)$  value is calculated. Thus, the processing order for the tasks does not affect the results of  $F(i, k)$  calculation.

3. For each task  $t_i$  in the RTL, calculate its metric  $\min_{F(i)}$ :

$$\min_{F(i)} = \min F(i, k) \quad \forall k \quad (5)$$

If there are tasks which satisfy  $\min_{F(i)} \geq BD_i$ , then we select the task which has the largest  $\min_{F(i)} - BD_i$  and assign it to the PE that corresponds to  $\min_{F(i)}$ .

4. On the other hand, if all the tasks in the RTL satisfy  $\min_{F(i)} < BD_i$ , then we select the next task to be scheduled and the PE it should be scheduled to as follows:

First, for each  $t_i$  in the RTL, a list  $L_i$  is generated which contains a list of PEs. Each PE  $p_k$  in that list must satisfy the condition  $F(i, k) \leq BD_i$ . More precisely, if task  $t_i$  is scheduled onto any PE in the list  $L_i$ , the finish time of  $t_i$  is guaranteed to meet its budget deadline  $BD_i$ .

Next, for each task  $t_i$ , let  $E_1^i$  be the minimum energy consumption if it is scheduled onto a tile belonging to the list  $L_i$ . And let  $E_2^i$  be the second minimum value for energy consumption<sup>3</sup>. A metric  $\delta_E^i = E_2^i - E_1^i$  is then calculated for each task.

Finally, we select the task in the RTL which has the largest  $\delta_E$  and this task is assigned to the PE which leads to  $E_1^i$  energy consumption.

---

<sup>3</sup>Note that when calculating  $E_1^i$  and  $E_2^i$ , the communication energy consumption is also taken into account since we have already scheduled all the sender tasks of task  $t_i$ .

- Update the schedule table of the corresponding PE and links. The level based scheduling is repeated until all the tasks are assigned and scheduled.

### Step 3. Search and repair

Since the optimization objective is the energy consumption minimization, the performance can not always satisfy the specified deadline constraints, although the budgeted deadline of each task does help the scheduler in meeting most of the deadlines. As shown by the experimental results in Sec. 6, if we just rely on aforementioned two steps, there can be occasional deadline misses. Because of this issue, we propose next a *search and repair procedure* which can be used to fix the missed deadlines. This procedure takes as input the currently generated schedule, and then iteratively improves the solution as described below.

The search and repair procedure has two main components: *local task swapping* (LTS) and *global task migration* (GTM). The relationship between LTS and GTM is shown in Fig. 6.

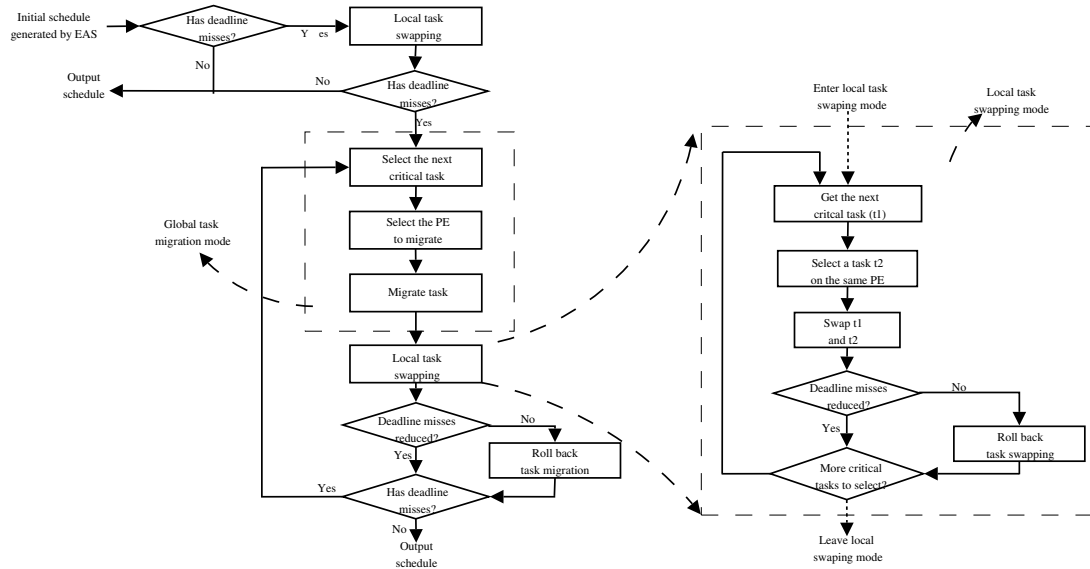


Figure 6: Flow of search and repair procedure described in Step 3 of the scheduling algorithm

In the LTS mode, the procedure will enumeratively pick up each critical task in the current solution and swap its order of execution with other non-critical tasks assigned to the same PE. The idea is to let the critical tasks execute earlier than non-critical tasks such that the deadline misses can be reduced. If the current swapping helps in reducing the deadline misses, then this move is accepted; otherwise, it is rejected. Obviously, since LTS only swaps the execution order of tasks on the same processing element, it cannot change the computation and communication energy consumption of the system.

In certain cases, however, the deadline misses can not be fixed by the LTS alone. This can be, for instance, the case when one PE is so heavily loaded that no matter how one

changes the tasks execution order on that PE, there will always be some tasks causing deadline misses (these tasks may not necessarily have a specified deadline, but they may cause one of the descendant tasks to miss its deadline). In this case, GTM helps by identifying a *critical* task and migrating it onto another PE. To reduce the impact of the GTM component on the energy increase, the destination PEs are tried in the increasing order of the execution and communication energy. If the migration of that task reduces the deadline misses, then the move is accepted. Otherwise, the next task will be selected for migration. Note that, as opposed to LTS, both communication energy consumption and computation energy consumption can change when applying GTM. Because of the greedy nature of this algorithm, the search and repair procedure will always converge.

## 6 Experimental Results

To evaluate its effectiveness, we implemented the above framework and performed several experiments on random task sets, embedded system benchmarks and a generic multimedia system.

### 6.1 Experiments on random benchmarks

Two categories of random benchmarks were generated using TGFF [9]. Each category contains 10 randomly generated benchmarks and each benchmark has around 500 tasks with about 1000 communication transactions. Both of these two categories of benchmarks are to be scheduled onto a  $4 \times 4$  heterogeneous NoC. To evaluate the robustness of our algorithm, various parameters are used in TGFF to generate benchmarks with different topologies and task/communication distributions. Compared to category I, benchmarks in category II have tighter deadlines.

We apply two versions of our algorithm to these benchmarks; that is, EAS-base (Energy-Aware Scheduler *without* search and repair) and EAS (Energy-Aware Scheduler *with* search and repair). To evaluate the energy savings of using our algorithm, we also implemented a standard *Earliest Deadline First* (EDF) scheduler and compared the schedules in terms of energy figures and deadline misses.

Fig. 7 and Fig. 8 show the comparison of energy consumption of the schedules generated for category I and II benchmarks, respectively. As shown in both figures, significant energy savings can be observed compared to the schedules generated by the EDF scheduler. For category I and category II of benchmarks, the results generated by the EDF consume, on average, 55% and 39% more energy compared to those generated by the EAS, respectively.

Although EAS-base is able to generate schedules for most of these benchmarks, the schedules it generates for benchmark 0 in Category I and benchmarks 0, 5, 6 in Category II fail to meet the specified deadlines. By post processing these schedules using the search and repair procedure in Section 5, EAS fixes all the deadline misses for these benchmarks with negligible increase in the energy consumption. However, applying the search and repair procedure does increase the run time of the scheduler. For the aforementioned four

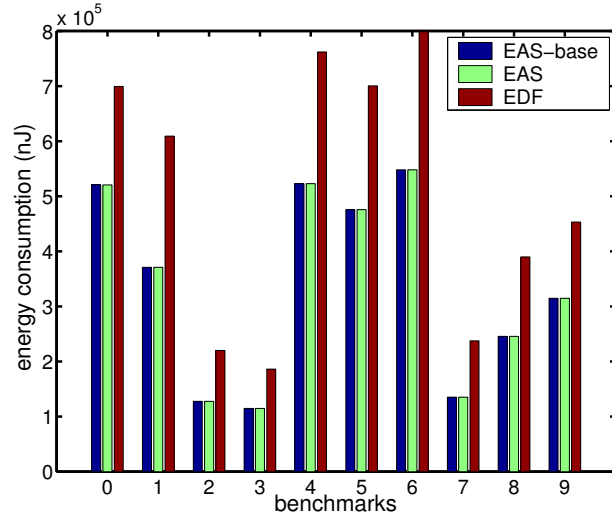


Figure 7: Energy consumption comparison using category I benchmarks

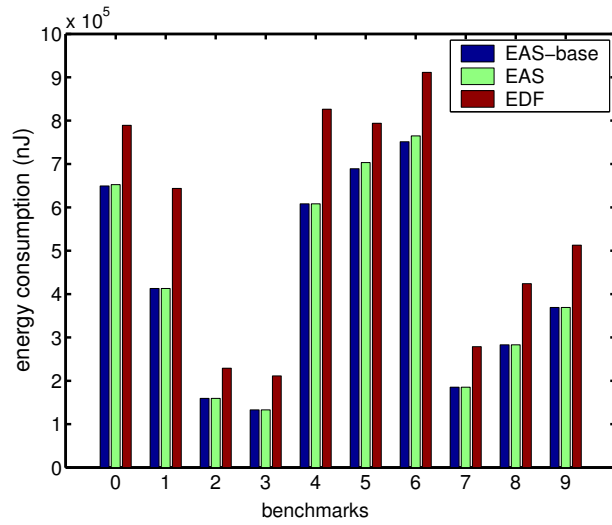


Figure 8: Energy consumption comparison using category II benchmarks

benchmarks, the run time increase from 1.77 sec., 2.45 sec., 3.23 sec. and 2.34 sec. to 2.17 sec., 214.21 sec., 1171.23 sec. and 958.95 sec., respectively.

## 6.2 Experiments on embedded system benchmarks

We also evaluated the performance of the proposed scheduling algorithm by applying it to some realistic embedded applications. Five benchmark applications are collected from *E3S* benchmark suite, namely *auto-indust*, *networking*, *telecom*, *consumer* and *office-automation* [23]. These benchmark applications contain 24 tasks, 13 tasks, 30 tasks, 12 tasks and 5 tasks, and are to be scheduled onto heterogeneous NoCs consisting of  $4 \times 4$  tiles,  $3 \times 3$  tiles,  $4 \times 4$  tiles,  $3 \times 3$  tiles and  $2 \times 2$  tiles, respectively. Table 1 reports the energy consumption of the scheduling results generated using the EAS scheduler and the EDF scheduler, respectively.

Table 1: Results on E3S benchmarks

E3S Benchmark	<i>auto-indust</i>	<i>consumer</i>	<i>network</i>	<i>office</i>	<i>telecom</i>
EAS Energy ( <i>nJ</i> )	167083	2.29285e+07	3.95295e+07	7.54779e+06	311710
EDF Energy ( <i>nJ</i> )	298190	2.57382e+07	4.62982e+07	3.95295e+07	Fail
Energy Savings (%)	44	10.9	14.6	80.9	—

As we can see from Table 1, significant energy savings can be achieved by using the proposed EAS algorithm compared to the scheduling solutions generated by EDF scheduler, without sacrificing the overall system performance. In fact, for the particular case of the *telecom* benchmark, it is interesting to note that the solution generated by EDF scheduler fails to meet the specified deadline because the greedy algorithm generates the final schedule in just one traverse. On the other hand, although the schedule generated by EAS-base algorithm also misses the deadline, with the help provided by the search and repair step, the solution generated by EAS is able to meet the specified the performance constraints.

## 6.3 Experiments on a multimedia system

To evaluate the potential of using our algorithm for real-world applications, we apply it to a set of generic Multimedia System Benchmarks (MSB).

The first system we consider consists of an MP3/H263 audio/video (A/V) encoder pair. We partitioned these two applications into 24 tasks, and inserted monitors in the C++ code to profile the intertask communication, as well as the execution time taken by each task. We experiment with three different real clips (*akiyo*, *foreman* and *toybox*). Using the profiled information in the application task graph, we schedule the task graph on a heterogeneous  $2 \times 2$  NoC architecture using EAS and the results are shown in Table 2. Also shown in Table 2 is the energy consumption of the scheduling results using the EDF scheduler.

Table 2: Results on an A/V encoder application

MSB Task Set	<i>akiyo</i>	<i>foreman</i>	<i>toybox</i>
EAS Energy ( $nJ$ )	56980	45390	55520
EDF Energy ( $nJ$ )	84078	52252	68405
Energy Savings (%)	32.2	13.1	18.8

The second system that we consider is an MP3/H263 A/V decoder system. The application is partitioned into 16 tasks and is profiled in the similar way as above. We schedule the task graph on a heterogeneous  $2 \times 2$  NoC and the results are shown in Table 3.

Table 3: Results on an A/V decoder application

MSB Task Set	<i>akiyo</i>	<i>foreman</i>	<i>toybox</i>
EAS Energy ( $nJ$ )	32504	27865	27486
EDF Energy ( $nJ$ )	38773	34231	33148
Energy Savings (%)	16.2	18.6	17.1

The last system that we consider integrates the above two systems by including an A/V encoder pair and an A/V decoder pair. The application contains 40 tasks and needs to be scheduled onto a heterogeneous  $3 \times 3$  tile-based NoC. The results are shown in Table 4.

Table 4: Results on A/V encoder/decoder application

MSB Task Set	<i>akiyo</i>	<i>foreman</i>	<i>toybox</i>
EAS Energy ( $nJ$ )	107648	78224	93413
EDF Energy ( $nJ$ )	153992	117553	129386
Energy Savings (%)	30.1	33.5	27.8

As we can see, compared to the schedule generated by EDF, significant energy savings can be achieved by using the EAS algorithm for all the above test cases. This further validates the effectiveness of our algorithm.

It is worth pointing out that these energy savings represent the *combined* effect of reducing both computation and communication energy values. For instance, with the movie clip *foreman*, the schedule generated using EAS successfully reduced the computation energy from 42674  $nJ$  to 34512  $nJ$ . In addition, it also reduced the communication energy from 74878  $nJ$  to 43710  $nJ$  by decreasing the average hops per packet from 2.55 to 1.70.

To see the trade-off between the energy savings and the performance constraints, we perform the following experiment on the integrated MSB application. Starting with a given encoding rate (40 frames/sec.) and decoding rate (67 frames/sec.), we slowly increase the required encoding rate and decoding rate and then observe its impact on the energy savings. The results are shown in Fig. 9.

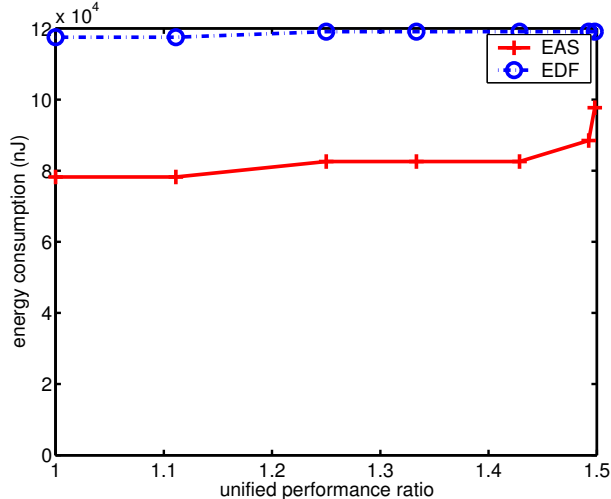


Figure 9: Performance and energy tradeoff for the integrated MSB benchmark

The X axis in Fig. 9 represents the required performance of the application compared to the baseline specification. For instance, 1.4 in the X axis means that the required encoding and decoding rates are  $1.4 \times 40 = 56$  frames/sec and  $1.4 \times 67 = 93.8$  frames/sec respectively. It is interesting to note that as the performance requirements become more stringent, the schedule generated by EAS consumes more energy consumption as the scheduler has less flexibility in assigning and ordering the execution of tasks/communication.

## 7 Conclusion and Future Work

In this paper, we have proposed an efficient energy-aware scheduling algorithm which statically schedules both communication transactions and computation tasks onto heterogeneous NoC architectures under real-time constraints.

Although we focus on the architectures interconnected by 2D mesh networks with XY routing schemes, our algorithm can be adapted to other regular architectures with different network topologies or different deterministic routing schemes. For instance, if the honeycomb topology in [4] is used, then we can still use Eq. (2) to calculate the  $E_{bit}$  metric for each sending and receiving PE pair, although this metric may no longer be determined by the *Manhattan* distance between them. Other practical implications of such changes will be explored as future work.

The proposed algorithm is targeted to real-time applications where the worst-case task execution time and inter-task communication volume are pre-characterized. It can also be applied to applications in the signal processing domain since the execution time of each task and the inter-task communication volume of such applications can usually be pre-determined. Thus, such applications can be modeled as CTGs which naturally expose the inherent inter-node parallelism in the application. Such a model allows partitioning and scheduling to be performed at compile time. On the other hand, due to the use of

such a *fully-static* scheduling paradigm, the algorithm can not be directly applied to those applications which include conditional branches. Extension of the algorithm to support such applications remains to be done as future work.

## References

- [1] W. J. Dally, B. Towles, "Route packets, not wires: on-chip interconnection networks," *Proc. DAC*, pp. 684–689, June 2001.
- [2] L. Benini, G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, pp. 70–78, Jan. 2002.
- [3] W. J. Dally, C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.
- [4] A. Hemani, *et al*, "Network on a chip: an architecture for billion transistor era," *Proc. of the IEEE NorChip Conf.*, Nov. 2000.
- [5] S. Kumar, *et al*, "A network on chip architecture and design methodology," *Proc. Symposium on VLSI*, pp. 117–124, April 2002.
- [6] Y. Zhang, X. Hu and D. Chen, "Task scheduling and voltage selection for energy minimization," *Proc. DAC*, June 2002.
- [7] J. Liang, S. Swaminathan and R. Tessier, "aSoC: a scalable, single-chip communications architecture," *Proc. Intl. Conf. on PACT*, Oct. 2000.
- [8] L. M. Ni, P. K. McKinley "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, Feb. 1993.
- [9] R. P. Dick, D. L. Rhodes and W. Wolf, "TGFF: task graphs for free," *Proc. Intl. Workshop on Hardware/Software Codesign*, March 1998.
- [10] E. Rijpkema, *et al*, "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip," *Proc. DATE*, March 2003.
- [11] E. Nilsson, "Design and implementation of a hot-potato switch in network on chip," M.S. thesis, Royal Institute of Technology (KTH), June 2002.
- [12] Y. Sun, "Simulation and performance evaluation for networks on chip," M.S. thesis, Royal Institute of Technology (KTH), December 2001.
- [13] P. Eles, *et al*, "Scheduling with bus access optimization for distributed embedded systems," *IEEE Tran. on VLSI*, vol. 8, No. 5, pp. 472–491, 2000.
- [14] G. C. Sih, E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Tran. on Parallel and Distributed Systems*, vol. 4, No. 2, 1993.
- [15] J. Luo, N. K. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," *Proc. ICCAD*, Nov. 2000.
- [16] T. T. Ye, L. Benini and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," *Proc. DAC*, June 2002.

- [17] J. Hu, R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints,” *Proc. ASP-DAC*, Jan. 2003.
- [18] J. Hu, R. Marculescu, “Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures,” *Proc. DATE*, March 2003.
- [19] S. Murali, G. De Micheli, “Bandwidth-constrained mapping of cores onto NoC architectures,” *Proc. DATE*, Feb. 2004.
- [20] T. T. Ye, L. Benini and G. De Micheli, “Packetized on-chip interconnect communication analysis for MPSoC,” *Proc. DATE*, March 2003.
- [21] H. Wang, *et al*, “Power model for routers: modeling Alpha 21364 and InfiniBand routers,” *IEEE Micro*, vol . 24, No. 1, pp. 26–35, Jan. 2003.
- [22] M. R. Garey, D. S. Johnson, “Computers and intractability: a guide to the theory of NP-completeness,” W.H. Freeman and Company, 1979.
- [23] R. Dick, “Embedded system synthesis benchmarks suites (E3S),” <http://helsinki.ee.princeton.edu/~dickrp/e3s>
- [24] C. J. Glass, L. M. Ni, “The turn model for adaptive routing,” *Proc. Intl. Symposium on Computer Architecture (ISCA)*, pp. 278–287, May 1992.
- [25] D. E. Culler, J. P. Singh and A. Gupta, “Parallel computer architecture, a hardware/software approach (second edition),” Morgan Kaufmann, 1996.
- [26] D. Roychowdhury, *et al*, “A voltage scheduling heuristic for real-time task graphs,” *Proc. Intl. Conf. on Dependable Systems and Networks*, June 2003.