Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints *

Jingcao Hu Radu Marculescu
Carnegie Mellon University
Pittsburgh, PA 15213-3890, USA
email: {jingcao, radum}@ece.cmu.edu

Abstract

In this paper, we present a novel Energy-Aware Scheduling (EAS) algorithm which statically schedules both communication transactions and computation tasks onto heterogeneous Network-on-Chip (NoC) architectures under real-time constraints. Our algorithm automatically assigns tasks onto different processing elements and then schedules their execution. At the same time, the algorithm also takes into consideration the exact communication delay by scheduling communication transactions in parallel. As the main 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 by using our energy-aware scheduler while meeting the specified performance constraints. For instance, for a complex multimedia application, 44% 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 to mitigate complex on-chip communication problems [1][2]. Such chips consist of regular tiles where each tile can be a general-purpose processor, a DSP, a memory subsystem, *etc* (e.g. [3][4][6]). A router is 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 also inevitably increases; this helps not only in optimizing performance for low power consumption, but also in keeping competitive design costs.

For embedded systems running on batteries, maximizing the battery-life is one of the chief design drivers. As such, targeting *low-energy consumption* is then extremely important. On the other hand, another essential characteristic of many embedded systems is their performance in terms of timing behavior; this is usually specified by *deadlines* associated with individual tasks.

For performance evaluation purposes, any application running on an embedded system can be described as a *Communication Task Graph* (CTG) (see right part of Fig. 1 for an example). The tasks in the CTG can have two types of dependencies: *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 to decide 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) leads to very different computation energy consumption.
- 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 topic for research, 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, in which a common objective is to minimize the energy consumption of the system 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 the communication congestion which may change dynamically throughout tasks execution. As we will show later, considering communication effects is critical for NoC architectures.

In this paper, we propose a new algorithm with the primary objective of scheduling both *communication and computation* for NoC architectures that minimizes the 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 contention, 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 are more accurate because

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

they take the effects of the traffic dynamics into consideration. 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 power and performance figures.

The remainder of this paper is organized as follows. We first review 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

Eles *et al* in [9] present a scheduling algorithm for distributed embedded systems which takes into consideration bus access optimization. Sih *et al* in [10] 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's energy consumption.

From another perspective, most of the work on low-power scheduling (e.g. [5][11]) focuses on architectures with dynamic voltage scaling or dynamic power management capabilities and tries to manipulate the task execution slacks to exploit them. The authors assume homogeneous, shared-bus architectures and thus, their work can not be applied in our scenario where the target architectures are 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 [1][3][4], as an illustrative platform for our work. In this section, we describe the regular tile-based architecture and the energy model for its communication network.

3.1 Architectural issues

The system under consideration is composed of $n \times n$ tiles interconnected by a $2D$ mesh network[1]. The left part of Fig. 1 shows the abstract view of the NoC architecture which has been used recently by several authors [1][4].

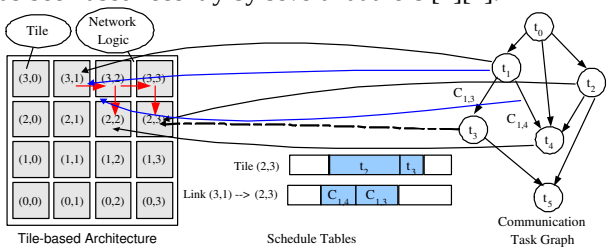

Figure 1. Tile-based NoC architecture

Each tile in Fig. 1 is composed of a *processing element* (PE) and a *router*. 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 performance characteristics, it takes different amounts of computation time and computation energy to execute the very same task on different tiles.

Due to the limited resources, the buffers in the routers are implemented using registers (typically in the size of one or two flits each). The routers use wormhole routing for packets delivery [7] which seems to be a reasonable solution for NoCs. A 5×5 crossbar switch is used as the switching fabric in the router [6][4]. For the sake of simplicity, the XY routing scheme is used to direct the packets across the chip. Note that, with small modifications, the algorithm can be applied to applications with other *deterministic* routing algorithms.

3.2 Energy modeling issues

Energy modeling for NoC architectures was a relatively unexplored area until recently. Ye *et al* in [12] 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 [13] and Ye *et al* in [14] suggest calculating E_{bit} as:

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

where $E_{S_{bit}}$ and $E_{L_{bit}}$ represent the energy consumed on the switch and on the link between tiles, respectively.

This is a very nice approximation since it eliminates the buffering energy consumption $E_{B_{bit}}$, which is a parameter tightly coupled with the network congestion whose accurate value can only be measured by time-consuming simulations. Thus, by using Eq. (1), the average energy consumption for sending one bit of data from tile t_i to tile t_j can be analytically calculated as:

$$E_{bit}^{t_i,t_j} = n_{hops} \times E_{S_{bit}} + (n_{hops} - 1) \times E_{L_{bit}} \quad (2)$$

where n_{hops} is the number of routers the bit passes on its way from t_i to t_j.

For $2D$ mesh networks with *minimal* routing, Eq. (2) shows that the average energy consumption of sending one bit of data from t_i to t_j is determined by the *Manhattan* distance between them. Although more accurate energy models exist in the literature (e.g. [15]), 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 for the optimization 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. For each task t_i, determining *which PE* in the NoC architecture should it be scheduled to, and *the time slot* when the task will be executed on that PE. For example in Fig. 1, this means to determine to which PE (e.g. (2,2), (2,3) *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 location (2,3), for instance, our approach should also determine which one should be executed first and when exactly it should be executed.

[1] We use $2D$ mesh network simply because it naturally fits the tile-based architecture. However, our algorithm can be extended for other topologies.

2. Determining the time slot for all the communication transactions in the application. For instance, if tasks t_1, t_3 and t_4 are assigned to PEs at locations (3,1), (2,2) and (2,3), respectively, then we need to determine the execution order of communication transaction $c_{1,3}$ and $c_{1,4}$ on the link from PE (3,1) to PE (3,2), 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 $r^i_j \in R^i$ 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 $e^i_j \in E^i$ 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 t_i and t_j. The direction 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 date to task t_j.

Obviously, to effectively exploit the parallelism provided by the NoC, the application has to be partitioned with the right granularity. The partition of 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 paper's focus is on communication and computation scheduling, in the following, 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 from 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 p_i to p_j, *i. e.*, E^{p_i,p_j}_{bit}. $b(r_{i,j})$ 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:

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

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 execution before or at their respective deadlines.

Since finding an optimal schedule for a multi-processor system that consumes the minimum energy is known to be NP-hard [16], 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 *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 $W_{t_i} = VAR_{e_i} \times VAR_{t_i}$. This weight is an important concept in our algorithm. Intuitively, the higher this weight, the higher the priority the task should have in selecting the PE to be scheduled onto (as the execution of that task has a higher impact on the energy consumption and the performance of the system).
3. Using M_{t_i} of different tasks, we calculate the slack for different paths, and then allocate the slack to different tasks based on their respective weights.

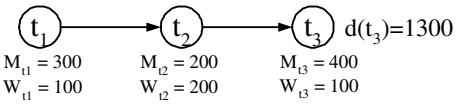

Figure 2. An example for budget slack allocation

For example, as shown in Fig. 2, 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. 2, 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 BD of task t_i.

Step 2. Level based scheduling

1. Generate the *Ready Tasks List*; that is, the tasks whose precedent tasks have already been scheduled. In the following, this list will be referred as 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 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. 3 is used to schedule the execution of these transactions.

```
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 3. The communication scheduler

In Fig. 3, the transactions in the LCT are first sorted by the 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 honoring the current schedule table of the path.
Please note that in this process, the communication transactions and tasks are only scheduled for the purpose of calculate $F(i,k)$. The schedule tables of both links and the PEs will be restored every time a $F(i,k)$ 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 $min_{F(i)}$:

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

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 satisfies $min_{F(i)} < BD_i$, then we use the following way to select the next task to be scheduled and to which PE it should be scheduled to.
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 $F(i,k) \leq BD_i$. Thus, this list gives all the PEs that if task t_i is scheduled onto any of the tiles, its budget deadline can be satisfied.
Next, for each task t_i, let E_1^i to be the minimum energy consumption if it is scheduled onto a tile in the list L_i. And E_2^i to be the second minimum energy consumption[2]. 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 δ_E and this task is assigned to the PE which leads to E_1^i energy consumption.
5. 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 for this scheduler is the energy consumption minimization, the performance can not be sometimes guaranteed to satisfy all 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, there can be occasional deadline misses if we just rely on aforementioned two steps. Because of this issue, here we present a *search and repair procedure* which can be used to fix the missed deadlines. This procedure takes as input the generated schedule, and then iteratively improves the solution in terms of deadline misses 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. 4.

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 so that the deadline misses can be reduced. If the 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 will *not* change the computation and communication energy consumption of the system.

In certain cases, the deadline misses can not be fixed by LTS. This can be, for instance, the case in which one PE is so heavily loaded that no matter how one changes the execution order of the tasks on that PE, there will always be some tasks causing deadline misses (these tasks may not necessarily have a specified deadline, but it causes one of its descendant tasks to miss its deadline). In this case, GTM helps by identifying a *critical* task and migrating it to other PEs. To reduce its impact on the energy increase, the destination PEs are tried in the increasing order of the execution and communication energy if that task is to be migrated onto the corresponding PEs. If the migration of that task reduces the deadline misses, then it's accepted. Otherwise, the next task will be selected for migration. Note that, as opposite to LTS, both communication energy consumption and computation energy consumption can be changed in GTM.

Because of the greedy nature of this algorithm, the search and repair procedure will always converge.

[2]Note that when we calculate 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 to t_i.

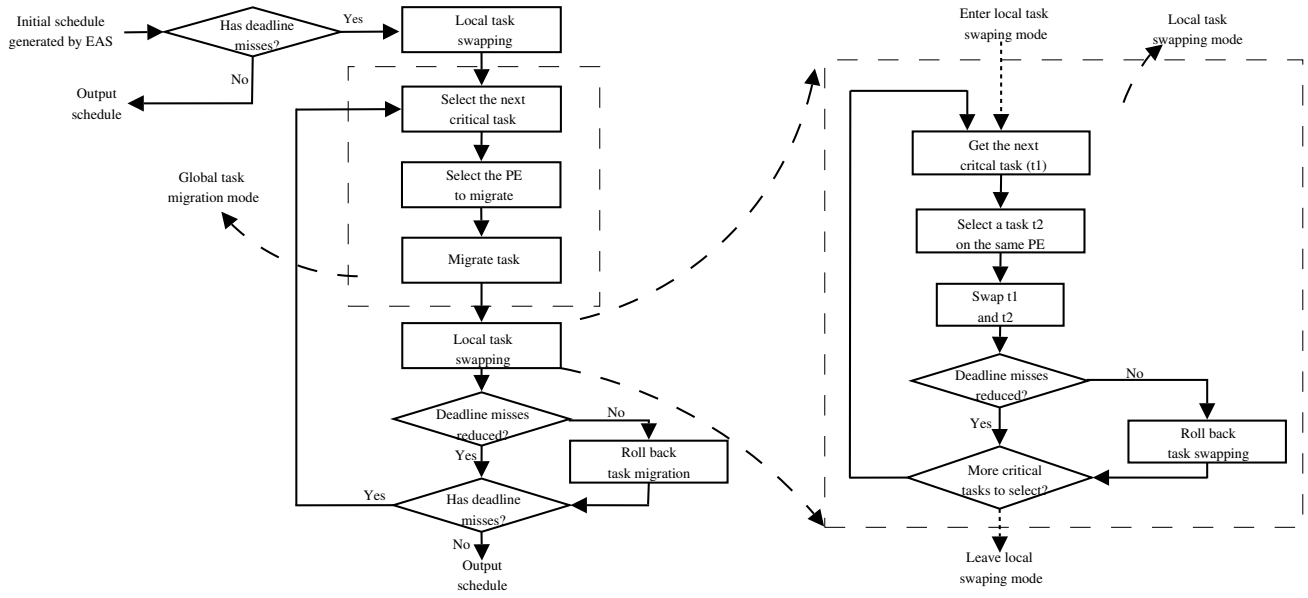

Figure 4. Flow of search and repair procedure

6 Experimental Results

To evaluate its effectiveness, we implemented the above framework and performed several experiments on random task sets and a set of generic multimedia systems.

6.1 Experiments on random benchmarks

Two categories of random benchmarks were generated using TGFF [8]. 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 4×4 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.

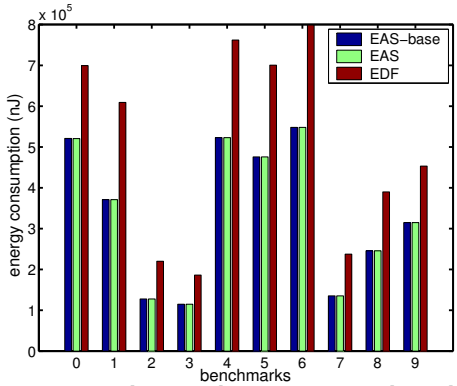

Figure 5. Comparison using category I benchmarks

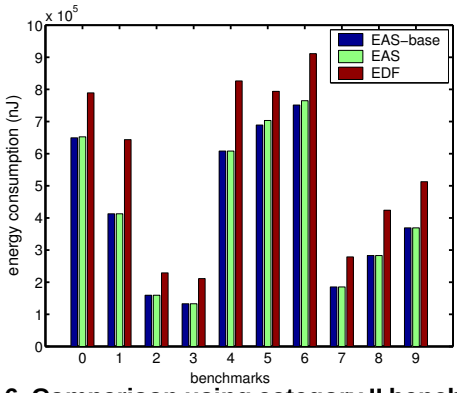

Figure 6. Comparison using category II benchmarks

Fig. 5 and Fig. 6 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 EDF consume, on average, 55% and 39% more energy compared to those generated by 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 benchmark 0, 5, 6 in Category II failed in meeting some of the specified deadlines. By post processing these schedules using search and repair procedure, EAS fixes all the deadline misses for all these benchmarks with negligible increase in the energy consumption. However, it does increase the run time of the scheduler. For the aforementioned four 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 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 partition these two applications into 24 tasks, and insert monitors in the C++ code to profile the intertask communication, as well as the execution time taken for 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 2×2 NoC architecture using EAS and the results are shown in Table 1. Also shown in Table 1 is the energy consumption of the scheduling results using the EDF scheduler.

Table 1. Results on an A/V encoder application

MSB Task Set	*akiyo*	*foreman*	*toybox*
EAS Energy (*nJ*)	56980	45390	55520
EDF Energy (*nJ*)	84078	52252	68405
Energy Savings (%)	47.6	15.1	23.2

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 2×2 NoC and the results are shown in Table 2.

Table 2. Results on an A/V decoder application

MSB Task Set	*akiyo*	*foreman*	*toybox*
EAS Energy (*nJ*)	32504	27865	27486
EDF Energy (*nJ*)	38773	34231	33148
Energy Savings (%)	19.3	22.8	20.6

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 3×3 tile-based NoC. The results are shown in Table 3.

Table 3. Results on A/V encoder/decoder application

MSB Task Set	*akiyo*	*foreman*	*toybox*
EAS Energy (*nJ*)	107648	78224	93413
EDF Energy (*nJ*)	153992	117553	129386
Energy Savings (%)	43.1	50.3	38.5

As we can see, compared to the schedule generated by EDF, significant energy savings can achieved by using our 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 are a *combined* effect of reducing both computation energy and communication energy. 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 reduces 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. 7.

The X axis in Fig. 7 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.

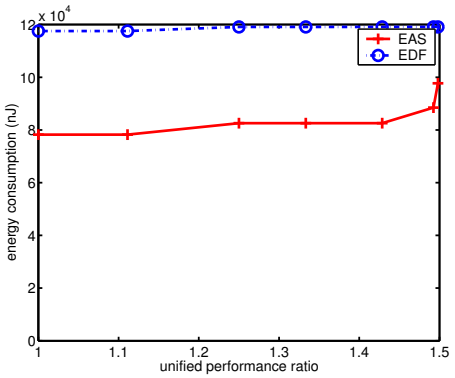

Figure 7. Performance and energy tradeoff

7 Conclusion and Future Work

In this paper, we 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 [3] 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 are planned to be explored 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] A. Hemani, *et al*, "Network on a chip: an architecture for billion transistor era," *Proc. of the IEEE NorChip Conf.*, Nov. 2000.

[4] S. Kumar, *et al*, "A network on chip architecture and design methodology," *Proc. Symposium on VLSI*, pp. 117–124, April 2002.

[5] Y. Zhang, X. Hu and D. Chen, "Task scheduling and voltage selection for energy minimization," *Proc. DAC*, June 2002.

[6] J. Liang, S. Swaminathan and R. Tesssier, "aSoC: a scalable, single-chip communications architecture," *Proc. Intl. Conf. on PACT*, Oct. 2000.

[7] L. M. Ni, P. K. McKinley "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, Feb. 1993.

[8] R. P. Dick, D. L. Rhodes and W. Wolf, "TGFF: task graphs for free," *Proc. Intl. Workshop on Hardware/Software Codesign*, March 1998.

[9] 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.

[10] 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.

[11] 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.

[12] T. T. Ye, L. Benini and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," *Proc. DAC*, June 2002.

[13] J. Hu, R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," *Proc. ASP-DAC*, Jan. 2003.

[14] T. T. Ye, L. Benini and G. De Micheli, "Packetized on-chip interconnect communication analysis for MPSoC," *Proc. DATE*, March 2003.

[15] 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.

[16] M. R. Garey, D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness," W.H. Freeman and Company, 1979.