J. Parallel Distrib. Comput. 96 (2016) 1–11

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Energy-efficient contention-aware application mapping and scheduling on NoC-based MPSoCs

Dawei Li [*], Jie Wu

Department of Computer and Information Sciences, Temple University, PA, 19122, United States

HIGHLIGHTS

- Energy-efficient application mapping and scheduling for Network-on-Chip systems.
- An approach that combines processor voltage scaling and link frequency tuning.
- A two-step method that provides clear and organized solution.
- Using genetic algorithm to achieve near-optimal voltage and frequency assignment.

ARTICLE INFO

Article history:
Received 17 September 2015
Received in revised form
16 March 2016
Accepted 13 April 2016
Available online 4 May 2016

Keywords:
Network-on-chip (NoC)
Application mapping
Energy-efficient scheduling
Dynamic voltage scaling
Dynamic link frequency tuning

ABSTRACT

We consider the problem of energy-efficient contention-aware application mapping and scheduling on Network-on-Chip (NoC) based multiprocessors. For an application represented by a directed acyclic graph, we present a model where voltage scaling techniques for processors can be combined with frequency tuning techniques for NoC links to save overall system energy consumption. We employ a two-step approach to solve the overall mapping and scheduling problem. First, the application mapping problem is formulated as a quadratic binary programming problem, which aims to minimize the communication energy; we apply a relaxation-based iterative rounding algorithm to solve it. With the mapping achieved, we further consider the application scheduling problem, which aims to find the optimal voltage level for each task of the application and optimal frequency level for each communication of the application to minimize the overall system energy consumption, given the application deadline. To attack the second problem, we first design an algorithm based on the earliest time first scheduling to determine the application's finish time if a voltage and frequency assignment is given; then, we develop a genetic algorithm to search the solution space for the voltage and frequency assignment that minimizes the overall system energy consumption and meets the application's deadline. Through these two steps, we produce a mapping and scheduling that meets the application's deadline, and significantly reduces the overall system energy consumption. Experiments are conducted for a number of randomly generated application graphs, as well as several real application graphs to verify the energy reduction and applicability of the proposed model and algorithms.

1. Introduction

MultiProcessor System-on-Chips (MPSoCs) play an important role in various computational systems, such as embedded systems, ad hoc and mobile devices, etc. Modern MPSoCs may consist of a large number of processors. Traditional bus-based on-chip communications are known to have poor scalability. Modern technology enables the use of Network-on-Chip (NoC) to support on-chip communications [2,8].

Energy consumption on these integrated systems has been a critical issue. Previously, energy-aware task scheduling problems have been studied extensively for traditional multiprocessor platforms without the consideration of communication time and energy. Generally speaking, processors are equipped with the capability of Dynamic Voltage and Frequency Scaling (DVFS) [4]. When the processor's utilization is low, it can be put to lower voltage/frequency levels to save energy consumption. For NoC-based MPSoCs, routers and links also consume a large portion of on-chip energy. The integrated routers and links of the Alpha 21 364 processor [21] consume 23 W out of the total chip power of 125 W (20%). Among the 23 W power, 58% of the power is consumed by the links; NoC links consume about 10% of the total

[*] Corresponding author.
E-mail addresses: dawei.li@temple.edu (D. Li), jiewu@temple.edu (J. Wu).

http://dx.doi.org/10.1016/j.jpdc.2016.04.006
0743-7315/© 2016 Elsevier Inc. All rights reserved.

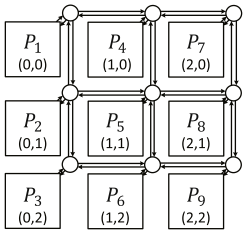

Fig. 1. A 3-by-3 mesh NoC architecture.

chip power. The IBM InfiniBand 8-port 12X switch is estimated to consume 31 W, where links take up about 20 W (65%) [23]. With the proposal of bufferless NoCs [14], links are expected to consume more power. Also, some research work focuses exactly on the power reduction on links [1]. Thus, taking into consideration the NoC energy consumption for application mapping and scheduling on NoC-based MPSoCs deserves extensive research endeavors.

The NoC architecture that we consider in this paper is a two-dimensional mesh. Fig. 1 shows an example. All processors of the MPSoC are assumed to be homogeneous [7,5], and are DVFS-enabled. The NoC links can also operate on different frequency levels, and thus, different power levels. The application can be represented by a Directed Acyclic Graph (DAG), as assumed by various existing works [10,17]. The problem we consider needs to address the following issues: (i) mapping the tasks of the application to the processors, (ii) setting a voltage level for each mapped task, or the corresponding processor, (iii) setting the frequency for each communication, or the links that construct the path of the communication, and (iv) deciding the order of task executions and task-to-task communications, such that tasks' and communications' precedence constraints are satisfied, and the contention of NoC link usage is explicitly avoided. Our final goal is to derive the mapping and scheduling that minimizes the overall system energy consumption, while the application's deadline is met.

1.1. Motivational example

We provide an example that motivates our research. Detailed assumptions and exact definitions are omitted here for simplicity. We consider the DAG application shown in Fig. 2(a). Fig. 2(b) and (c) show two possible mappings of the application. XY minimal routing [30] is adopted for its simplicity. We assume that each link's energy consumption during one unit of communication time is one unit. Then, the link energy consumption of mapping 1 can be calculated as follows: $10 + 20 + 15 \times 2 = 60$. 15 is multiplied by 2, because the communication, $A \rightarrow C$ should traverse 2 links. Similarly, the link energy consumption of mapping 2 can be calculated as $10 \times 2 + 20 + 15 = 55$; compared to mapping 1, 5 units of link energy is saved. Notice that we just calculated energy consumption on links; actually, similar calculations reveal that the energy consumption on routers of mapping 2 is also less then that of mapping 1. To save energy, mapping 2 should be adopted. The intuition is that the tasks that have greater communication volumes should be mapped to closer processors.

Given mapping 2, we now consider the actual scheduling of the application, with both the precedence constraints and the usage of NoC links in mind. Task A can begin executing first. Since communications $A \rightarrow C$ and $A \rightarrow B$ both need to use the link from A to C, they cannot occur at the same time; in other words, they must be serialized. In this situation, we have two strategies. One is to schedule communication $A \rightarrow C$ first; the second one is to schedule communication $A \rightarrow B$ first.

In the first strategy, after communication $A \rightarrow C$, only communication $A \rightarrow B$ can happen, because B and C have not received all the data they need. After that, B can start executing, followed by communication $B \rightarrow C$. Finally, task C executes. The overall scheduling of the first strategy is shown in Fig. 2(d); the schedule length, or the last task's finish time, is 85.

In the second strategy, after communication $A \rightarrow B$, B can start executing because it will have received the data that it needs from A; at the same time, communication $A \rightarrow C$ can also begin because the links that this communication requires are available now. Thus, task B's execution and communication $A \rightarrow C$ can happen simultaneously. Communication $B \rightarrow C$ can begin when task B finishes its execution. Finally, C can execute. The overall scheduling of the second strategy is shown in Fig. 2(e); the schedule length is 70.

Assume that the deadline of this application is 85. In scheduling 1 (Fig. 2(d)), though processors and links are capable of operating on variable voltage and frequency levels, respectively, they cannot do so. This is because they are all operating at the highest levels, and reducing any of them will increase the application's schedule

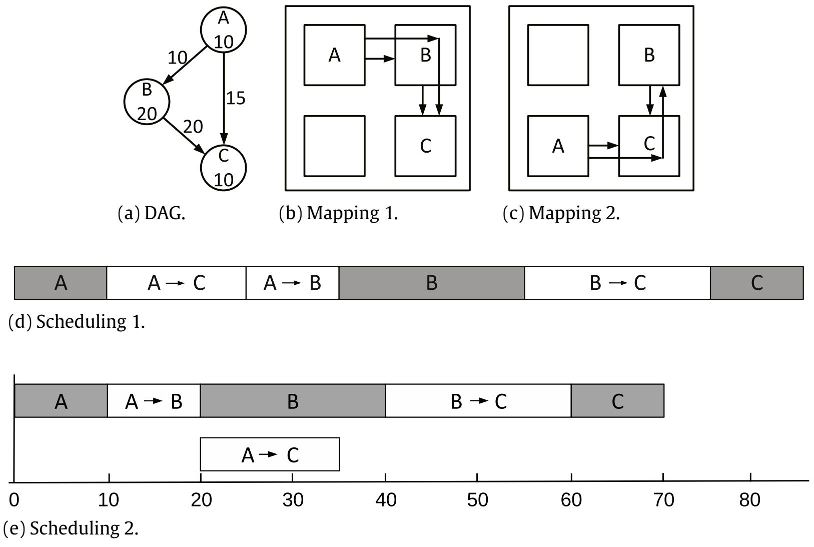

Fig. 2. Motivational example. (a): The application consists of three tasks A, B, and C, with execution times of 10, 20, and 10, respectively. The communication durations of $A \rightarrow B$, $B \rightarrow C$, and $A \rightarrow C$ are 10, 20, and 15, respectively. All these values are obtained when the processors operate at their highest voltage levels and links operate at their highest frequencies. (b) and (c): Two possible mappings of the application. (d) and (e): Two possible schedulings based on mapping 2; we use dark blocks to represent computation duration, and white blocks to represent communication durations; the two figures share the same time axis.

length, and will result in the application missing its deadline. However, in scheduling 2 (Fig. 2(e)), both tasks are allowed to operate on lower voltage levels, and links are allowed to operate on lower frequency levels to save energy. Thus, we can further apply voltage scaling and frequency tuning techniques to save energy consumption. Notice that we have many options. For example, we can reduce just the voltage of task B, or both A and B, or A, B, and C at the same time, as long as the deadline is still met; we can also choose to what extent to scale the voltage of each task. We can also choose to reduce the frequencies of communications $A \rightarrow B$, $B \rightarrow C$, and/or $A \rightarrow C$. Among various choices, we need to decide which choice results in the least overall energy consumption.

In this paper, we address the problem of application mapping and scheduling, while considering all of the four issues as described. We apply a two-step approach to solve the overall problem. Specifically, the application mapping problem addresses the first issue, and the application scheduling problem addresses issues ii–iv. The organization of the paper is as follows. Related works and our contributions are described in Section 2. The system models, which include the application model, the platform model, and some basic assumptions, are provided in Section 3; the formal problem definition is also presented. Our two-step approach is discussed in detail in Sections 4 and 5: Section 4 addresses the application mapping problem; Section 5 addresses the application scheduling problem. Simulations are presented in Section 6. Conclusions are made in Section 7.

2. Related works and our contributions

A comprehensive survey for application mapping techniques on NoCs is provided by Sahu et al. [22]. A two-step Genetic Algorithm (GA) for application mapping onto NoCs has been proposed in [17]. [31] provides a delay model considering practical factors, and also applies the genetic algorithm. However, the goal of [17,31] is to reduce the overall execution time, and they do not consider energy issues. Authors in [20] consider designing heterogeneous on chip networks according to applications' bandwidth and latency sensitiveness.

Power/energy management is an important issue in NoC-based MPSoCs [9,28]. [12] addresses energy-aware task allocation for NoC-based heterogeneous multiprocessors. DVFS-enabled processors are assumed; operating frequencies of network links are assumed to be fixed. Similar models, as that of [12], are also considered in [3,11]. [29] utilizes the *network slack* to adjust the operating frequencies of routers to provide *just enough power* to NoC to meet the deadlines. Contention-aware application mapping is addressed by [6], which quantifies the contention and aims to minimize it along with weighted distance. [23] proposes voltage and frequency scaling with links based on a link's average utilization for energy optimization.

Our work differs from existing works in two important aspects. First, we address the problem of both application mapping and application scheduling, where NoC link usage contentions are considered explicitly. Second, we present a model, where DVFS techniques for processors and frequency tuning techniques for NoC links can be combined together to achieve the goal of minimizing system energy consumption; in other words, the communication and/or computation slacks of an application can be shared efficiently by both communications and computations to achieve overall energy reduction.

Considering application mapping and scheduling in one step is two complex. To attack the problem, we employ a two-step approach. First, we formulate the application mapping problem as a Quadratic Integer Programming (QIP), more precisely, a Quadratic Binary Programming (QBP) problem, and apply a relaxation-based iterative rounding technique to solve it. Second, after achieving the mapping, we further consider the energy-efficient scheduling problem, which aims to find the optimal voltage setting for each task of the application and optimal frequency setting for each communication of the application to minimize the overall system energy consumption. The DAG scheduling problem to find the minimum schedule length is generally NP-hard [24]. Thus, we adopt another two-step approach to solve our energy-efficient scheduling problem: first, we design an algorithm based on the Earliest Time First (ETF) scheduling to get the application's earliest finish time, given a voltage and frequency assignment; next, we develop a genetic algorithm to search the solution space (various voltage and frequency assignments) to find a voltage and frequency assignment that tries to minimize the overall system energy consumption, and meets the application deadline.

3. System model and problem definition

3.1. Application model

An application can be represented by a DAG, $G = (T, E)$. $T = \{\tau_1, \tau_2, \dots, \tau_N\}$ is the task set of the application. $N = |T|$ is the number of tasks. Each task has an execution requirement of c_i, which represents the number of processor clock cycles. $E = \{e_1, e_2, \dots, e_{|E|}\}$ is communication set of the application, where $e_i = (\tau_s, \tau_d, w_i)$, $(s \neq d, \tau_s, \tau_d \in T)$ represents the communication from task τ_s to task τ_d. w_i is the communication volume of e_i in the unit of bit. $|E|$ is the cardinality of set E, in other words, the total number of communications of all the tasks. Communications also represent data dependencies. A task can begin to execute if and only if it has received all the data destined to it [10,17].

3.2. Platform model

3.2.1. Processor model

We assume homogeneous independent processors. The processors are DVFS enabled, and can operate on a set of voltage levels, $v = \{v_1, v_2, \dots, v_{n_V}\}$, where n_V is the total number of voltage levels. Without loss of generality, we assume these voltage levels are sorted in ascending order. Denote the execution speed (or the clock frequency) and the power consumption of a processor operating at voltage v_j by f_j^v and p_j^v, respectively.

We assume that one processor can hold at most one task, due to its limited resources, such as cache, memory, etc. [6]. When task τ_i, $(1 \le i \le N)$ is executed by a processor operating at voltage v_j, $(1 \le j \le n_V)$, its execution time is

$$t_{i,j}^v = c_i / f_j^v, \tag{1}$$

and its energy consumption is

$$\mathcal{E}_{i,j}^v = p_j^v c_i / f_j^v. \tag{2}$$

Though exact relations between v_j and f^{v_j}, p^{v_j} may exist, we do not make such assumptions in our model. What we are interested in are the discrete frequency points and the corresponding power consumption values. Thus, various processor models can be applied; and other practical considerations can be easily incorporated into our model, such as the static power consumption of processors; also, the voltage switching overhead, if not significantly influential, can be taken into account by regarding it as a fixed power consumption value at the given frequency point.

3.2.2. NoC model

The dimension of the NoC architecture is $n_R \times n_C$, where n_R is the number of rows, and n_C is the number of columns. Each processor is associated with a router. The total numbers of processors and routers are both $M = n_R n_C$. A router has at most five ports, each for communicating with the associated processor and its at most four neighbor routers. The ports for communicating with neighbor routers and its associated processor are equipped with two directed links with two opposite directions, namely, one from itself, and the other to itself. We call the links connecting routers *global* links, and the links connecting a router and a processor are called *local* links. We assume that all global links are identical with the same bit width b_w, and have only one virtual channel. Local links have four virtual channels, and the associated processor can send through the four virtual channels and/or receive from the four virtual channels at the same time. Similar assumptions are made in [17]. We neglect energy consumption on local links, and only consider the energy consumption on global links. In the rest of the paper, all links refer to global links, unless otherwise specified.

All the links can operate at a set of frequencies, $f = \{f_1, f_2, \ldots, f_{n_F}\}$, where n_F is the total number of available frequencies. Like the voltage levels for processors, the frequencies are also in ascending order. The bandwidth of a global link operating at frequency f_k is $B^f_k = b_w f_k$. We refer to the timing model in [19], and assume that the communication volumes are large enough such that the serialization delay dominates the router delay. For a communication that needs to traverse several links, the link with the lowest frequency determines the serialization delay. Thus, to avoid unnecessary energy waste, for each communication, the operating frequencies of all its traversed links should be kept the same as the lowest one among them. We denote this frequency as the corresponding communication's frequency. The time for communication e_i, if it is carried by links operating at f_k, can be calculated as follows:

$$t^f_{i,k} = w_i / B^f_k = w_i/(b_w f_k), \tag{3}$$

where f_k is the communication e_i's frequency.

We adopt the bit energy model presented in [27,19]. The bit energy consumption of communication e_i can be calculated as follows: $\mathcal{E}_{bit_i} = (d_i + 1)\mathcal{E}_{Rbit} + d_i \mathcal{E}_{Lbit}(f)$. d_i is the Manhattan distance of the communicating tasks when they have been mapped to the processors, and thus, is equal to number of global links that the bit passes. $(d_i + 1)$ equals the number of routers that a bit passes during the communication. $\mathcal{E}_{Rbit}$ is the energy consumption of a bit on one router, and $\mathcal{E}_{Lbit}(f)$ is the energy consumption of a bit on one link when all the links of e_i operate at frequency f. Denote the power consumption of a link at frequency f_k by p^f_k. Then, $\mathcal{E}_{Lbit}(f_k) = p^f_k/(b_w f_k)$. Thus, when communication e_i, $(1 \le i \le |E|)$ is routed by links operating at frequency level f_k, $(1 \le k \le n_F)$, the energy consumption of e_i can be calculated as follows:

$$\mathcal{E}^f_{i,k} = w_i((d_i + 1)\mathcal{E}_{Rbit} + d_i p^f_k/(b_w f_k)), \tag{4}$$

where w_i is the communication volume of e_i. Again, though an exact relation between f_k and p^f_k may exist, we do not make such an assumption; thus, any power consumption model can be applied here, and other practical considerations can be easily incorporated into our model. Though our frequency tuning is communication-by-communication and link-by-link, the frequency changing rate is not as high as we may imagine. As a matter of fact, we schedule communications that share the same link(s) serially; thus, a link's frequency is only changed when it sees a new communication. Link frequency tuning at this level is quite common [23].

3.3. Additional assumptions

In this paper, we do not consider adjusting routers' frequency, because several communications traversing the same router(s) will make the problem more complex. We just assume that the router's frequency are fixed and can satisfy the clock frequencies of all links. Adjusting routers' frequency is left for future work. We assume the winner-take-all bandwidth allocation on NoC links, which allocates all of the bandwidth to one packet until it is finished, before serving any other packet [17]. Thus, at any time, a link can only serve one packet transmission; if two communications of the application graph intend to use the same link(s) for routing at the same time, they must be serialized due to contention. We assume wormhole routing, for it reduces the requirement for buffer. XY routing is used for simplicity.

3.4. Problem definition

We are given a mesh NoC-based MPSoC, where homogeneous independent processors can operate at a set of voltage levels, and NoC links can operate at a set of frequency levels. We are also given an application represented by a DAG, which consists of tasks with data dependencies and task-to-task communications. Our goal is to derive a task-to-processor mapping, as well as a scheduling such that the application's deadline is met, all the precedence constraints are satisfied, contentions of NoC link usage are explicitly avoided, and the overall system energy consumption is minimized.

Taking both mapping and scheduling into account at the same time makes the problem too complex to solve. We employ a two-step approach to attack the overall problem. We first deal with the mapping problem, which aims at minimizing the communication energy, assuming all links are operating at the highest frequency level. We formulate the mapping problem as a QBP problem and apply a relaxation-based iterative rounding algorithm to solve it. With the mapping achieved, the second step tries to derive an energy-efficient scheduling that takes into account the voltage and frequency assignments for tasks and communications, as well as deciding the execution order of all tasks and communications.

4. Energy-efficient application mapping

4.1. Problem analysis

For the application mapping problem, we aim to minimize the total energy consumption for communications, i.e., links' energy consumption and routers' energy consumption, assuming that all links are operating at the highest frequency. The total energy consumption for all communications can be calculated as follows:

$$\begin{aligned}\mathcal{E}^f_{total} &= \sum_{i=1}^{|E|}\{w_i d_i(\mathcal{E}_{Rbit} + \mathcal{E}_{Lbit}(f_{n_F})) + w_i\mathcal{E}_{Rbit}\}\\ &= (\mathcal{E}_{Rbit} + \mathcal{E}_{Lbit}(f_{n_F}))\sum_{i=1}^{|E|} w_i d_i + \sum_{i=1}^{|E|} w_i \mathcal{E}_{Rbit}.\end{aligned} \tag{5}$$

When the application is given, w_is have known values; $\mathcal{E}_{Rbit}$, $\mathcal{E}_{Lbit}(f_{n_F})$ are parameters related to the platform. Finding a mapping that minimizes $\mathcal{E}^f_{total}$ is equivalent to finding a mapping that minimizes $\sum_{i=1}^{|E|} w_i d_i$, i.e., the sum of weighted distances of all communications.

We denote X as a mapping matrix. $X_{i,j} = 1$ means that task τ_i is mapped to processor j, $\forall i = 1, 2, \ldots, N$, $\forall j = 1, 2, \ldots, M$; otherwise, $X_{i,j} = 0$. We further denote that $X_i = (X_{i,1}, X_{i,2}, \ldots, X_{i,M})$.

Denote L as the minimal/Manhattan distance matrix between processors. Specifically, $L_{i,j}$ is the Manhattan distance between processors i and j, $\forall i, j = 1, 2, \dots, M$. This matrix can be easily calculated by first conducting an index-to-coordinates conversion, and then summing the absolute values of the differences of the x-axes and y-axes of processors i and j. For example, the coordinates of processor i in the mesh are $(\lfloor (i-1)/n_R \rfloor, i-1-\lfloor (i-1)/n_R \rfloor n_R)$; the coordinates of processor j in the mesh are $(\lfloor (j-1)/n_R \rfloor, j-1-\lfloor (j-1)/n_R \rfloor n_R)$. $L_{i,j}$ can be calculated as follows:

$$L_{i,j} = \left| \left\lfloor \frac{i-1}{n_R} \right\rfloor - \left\lfloor \frac{j-1}{n_R} \right\rfloor \right| + \left| i - j - \left(\left\lfloor \frac{i-1}{n_R} \right\rfloor - \left\lfloor \frac{j-1}{n_R} \right\rfloor \right) n_R \right|. \quad (6)$$

We use the following lemma to calculate the distance between two tasks given a mapping.

Lemma 1. *The minimal/Manhattan distance between τ_i and τ_j can be calculated as $X_i L X_j^T$, $\forall i, j = 1, 2, \dots, N$.*

Proof. The distance between the ith and jth processors can be calculated by ALB, where $A = (a_1, \dots, a_M)$, $a_k = 0, \forall k \neq i, a_i = 1$, and $B = (b_1, \dots, b_M)^T$, $b_k = 0, \forall k \neq j, b_j = 1$. Assume that τ_i is mapped to processor k; then, $X_{i,k} = 1$; also, assume that τ_j is mapped to processor l; then, $X_{j,l} = 1$. All other values of X_i and X_j are zeros. Then, the distance between tasks τ_i and τ_j is just the distance between the kth and the lth processors, and can be calculated by $X_i L X_j^T$, $\forall i, j = 1, 2, \dots, N$, where X_j^T is the transpose of X_j.

Thus, the following $M \times M$ matrix represents the distances between all task pairs given a mapping X:

$$L' = XLX^T, \quad (7)$$

where

$$L'_{i,j} = X_i L X_j^T. \quad (8)$$

Denote the tasks' communications by an $N \times N$ matrix, $\mathcal{C}_{N\times N}$. If there is no direct communication from τ_i to τ_j, $\mathcal{C}_{i,j} = 0$. If there exists a direct communication from tasks τ_i to τ_j, $\mathcal{C}_{i,j}$ equals the volume of this communication. Thus, the weighted distance from τ_i to τ_j can be calculated as $L'_{i,j}\mathcal{C}_{i,j}$. To minimize the total weighted distance of all communications is equivalent to minimize $\sum_{i=1}^{N} \sum_{j=1}^{N} L'_{i,j}\mathcal{C}_{i,j}$. The application mapping problem can be formulated as follows:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} L'_{i,j}\mathcal{C}_{i,j} \quad (9)$$

$$\text{s.t.} \sum_{j=1}^{M} X_{i,j} = 1, \quad \forall i = 1, 2, \dots, N; \quad (10)$$

$$\sum_{i=1}^{N} X_{i,j} \leq 1, \quad \forall j = 1, 2, \dots, M; \quad (11)$$

$$X_{i,j} = 0 \text{ or } 1, \quad \forall i = 1, 2, \dots, N, \ \forall j = 1, 2 \dots, M. \quad (12)$$

The constraints in (10) mean that each task is assigned to a processor. The constraints in (11) mean that a processor can be assigned at most one task. By further transformation, we notice that the problem formulated in (9)–(12) is a standard Integer Quadratic Programming (IQP) problem, more precisely, a Binary Quadratic Programming (BQP) problem, which is known to be NP-hard [15].

Algorithm 1 RIRAM: Relaxation-based Iterative Rounding for Application Mapping

```
Input: The task set $T = \{\tau_1, \tau_2, \cdots, \tau_N\}$ and associated communication matrix $\mathcal{C}_{N\times N}$;
Output: Binary matrix $Map_{N\times M}$ indicating the final mapping;
1: Initialize the assignment matrix: $Map_{i,j} = 0$ ($\forall i = 1, 2, \cdots, N; j = 1, 2, \cdots, M$);
2: Sort the task, such that $\mathcal{V}_{i_1} \geq \mathcal{V}_{i_2} \geq \cdots \geq \mathcal{V}_{i_N}$.
3: for $k := 1$ to $N$ do
4:     Solve Prob $k$.
5:     $X_{i_k,j_k^*} = \max\{X_{i_k,1}, X_{i_k,2}, \cdots, X_{i_k,M}\}$.
6:     $X_{i_k,j_k^*} = 1$; $X_{i_k,j} = 0, \forall j \neq j_k^*$.
7:     $Map_{i_k,j_k^*} = 1$;
   return $Map$;
```

4.2. Algorithm

To solve the problem, we propose to use a relaxation-based iterative rounding method [18]. The basic idea of this method is as follows. We first relax the constraints in (12), such that $X_{i,j}$ can be any fraction between 0 and 1, i.e., the constraints in (12) are replaced by

$$0 \leq X_{i,j} \leq 1, \quad \forall i = 1, 2, \dots, N, \ \forall j = 1, 2 \dots, M. \quad (13)$$

We denote the relaxed problem by *Prob* 1. Notice that *Prob* 1 is a convex quadratic programming problem that can be solved efficiently by several methods, such as the Active Set method and the Interior Point method. Intuitively, when mapping tasks to processors, the task that has the greatest communication volume (including both the communications sourced from it and destined at it) has the highest priority, and should be considered first. Thus, we sort the tasks in the non-increasing order of the following value:

$$\mathcal{V}_i = \sum_{k=1}^{N} \mathcal{C}_{i,k} + \sum_{l=1}^{N} \mathcal{C}_{l,i}. \quad (14)$$

Denote the sorted tasks as $(\tau_{i_1}, \tau_{i_2}, \dots, \tau_{i_N})$, where $(i_1, i_2, \dots, i_N)$ is a permutation of $(1, 2, \dots, N)$, and $\mathcal{V}_{i_1} \geq \mathcal{V}_{i_2} \geq \cdots \geq \mathcal{V}_{i_N}$.

The next step is to assign τ_{i_1} based on the solution of *Prob* 1. We also denote the optimal solution for *Prob* 1 as $X_{N\times M}$ without any confusion. We find the j_1^* such that X_{i_1,j_1^*} is the greatest among all $X_{i_1,j}$ values, $\forall j = 1, 2, \dots, M$. Then, we map τ_{i_1} to processor j_1^*, which means that, we set $X_{i_1,j_1^*} = 1$, and $X_{i_1,j} = 0, \forall j \neq j_1^*$. Then we update *Prob* 1 as

$$\min \sum_{i=1}^{N} \sum_{j=1}^{N} L'_{i,j}\mathcal{C}_{i,j} \quad (15)$$

$$\text{s.t.} \sum_{j=1}^{M} X_{i,j} = 1, \quad \forall i = 1, \dots, i_1 - 1, \ i_1 + 1, \dots, N; \quad (16)$$

$$\sum_{i=1}^{N} X_{i,j} \leq 1, \quad \forall j = 1, \dots, M; \quad (17)$$

$$0 \leq X_{i,j} \leq 1, \quad \forall i = 1, \dots, i_1 - 1, \ i_1 + 1, \dots, N, \ j = 1, \dots, M. \quad (18)$$

which we denote by *Prob* 2, since the solution of it will provide the information for mapping τ_{i_2}. Though *Prob* 2 and *Prob* 1 look similar, they are actually quite different. In *Prob* 2, $X_{i_1,1}, \dots, X_{i_1,M}$ have fixed values, and the optimization variables are just $X_{1,1}, \dots, X_{1,M}$; $X_{2,1}, \dots, X_{2,M}$; $\dots$; $X_{i_1-1,1}, \dots, X_{i_1-1,M}$; $X_{i_1+1,1}, \dots, X_{i_1+1,M}$; $\dots$; $X_{N,1}, \dots, X_{N,M}$.

To map τ_{i_2}, the similar approach is applied. Namely, we first solve *Prob* 2, and find the X_{i_2,j_2^*} that is the greatest among all $X_{i_2,j}$ values, $\forall j = 1, 2, \ldots, M$. Then, map τ_{i_2} to processor j_2^*; in other words, set $X_{i_2,j_2^*} = 1$, and $X_{i_2,j} = 0,\ \forall j \neq j_2^*$.

After that, update *Prob* 2 as *Prob* 3, and map τ_{i_3} based on $X_{i_3,j}$ (solved for *Prob* 3), $\forall j = 1, 2, \ldots, M; \ldots$; update *Prob* $(k-1)$ as *Prob* k, and map τ_{i_k} based on $X_{i_k,j}$ (solved for *Prob* k), $\forall j = 1, 2, \ldots, M; \ldots$. We conduct the above iterative rounding and mapping till we have mapped all of the tasks.

The overall algorithm is outlined in Algorithm 1, which involves solving the convex optimization problems N times. In every iteration, we consider mapping the task with the most traffic volume with other tasks; after that we update our convex programming problem and consider the next unassigned task with the most traffic volume with other tasks, taking into account that all previously considered tasks have been mapped to a processor. Since solving the convex optimization problems are polynomial, Algorithm 1 is still a polynomial time algorithm.

5. Energy-efficient contention-aware application scheduling

5.1. Problem analysis

With the mapping achieved, the next step is to derive a scheduling for the mapped application. A scheduling needs to consider all of the following issues: set the voltage levels for all tasks; set the link frequencies for all communications; satisfy the precedence constraints of the application graph; decide the overall order and start times and finish times of tasks and communications.

Since the application consists of two kinds of events: tasks and communications, to enable the use of traditional DAG scheduling algorithms, we first transform the application graph to an extended one, in which each unified node represents an event that can be either a task or a communication. The transformation is quite simple. First construct an extra node for each edge/communication in the original graph; we call this node a communication node; each communication node has one and only one parent, which is the source of the communication, and has one and only one child, which is the destination of the communication. The nodes in the original graph are kept unchanged, and they are called task nodes. The execution time of each node is the task's execution time or a communication's time. We denote the transformed graph by $G^* = (T^* + E^*, E')$, where T^* is the set of task nodes, E^* is the set of communication nodes, and E' is the set of edges in the transformed graph.

If the deadline of the application is infinite, all tasks and communications can choose the lowest voltage and frequency available. However, for a practical application with a deadline constraint, choosing the lowest voltage and frequencies will result in the application missing its deadline. Given a voltage and frequency assignment for the application, the energy consumption of the application can be calculated, which can be achieved by summing up all the energy consumption of tasks and communications. To make this energy consumption value practically achievable, we should derive a valid scheduling that utilizes these voltage and frequency settings, namely, a scheduling with a schedule length that is less than or equal to the deadline of the application.

However, the DAG scheduling problem that tries to find the minimum schedule length is generally NP-hard [24]. Involving the voltage and frequency assignment will make the scheduling problem more complex. Thus, we adopt another two-step approach: first, we design an algorithm based on the Earliest Time First (ETF) scheduling to get the application's finish time, given a voltage and frequency assignment; next, we develop a genetic algorithm to search the solution space to find a voltage and frequency assignment that tries to minimize the overall system energy consumption, and meets the application deadline.

Algorithm 2 ETFGBF

```
Input: $G = (T, E)$ and a voltage and frequency assignment;
1:  $\mathcal{E}$ = energy consumption of all tasks and communications;
2:  Conduct graph transformation to get the extended graph $G^* = (T^* + E^*, E')$; calculate the execution time and b-level of each unified node in $T^* + E^*$; initialize the ready times of entry nodes as zeros;
3:  $Unscheduled = |T^*| + |E^*|$;
4:  while $Unscheduled \neq 0$ do
5:      $ReadySet$ = nodes in $G^*$ with all parents scheduled;
6:      Choose the node in $ReadySet$ with the earliest ready time to schedule; when encountering a tie, choose the one with the greatest b-level to schedule; if encountering a tie again, choose randomly;
7:      For any node that is a child of the just scheduled node, label this parent as scheduled; if all its parents have been scheduled, label it as ready, and set the node's ready time as the maximum finish time of all its parents;
8:      For any node that has potential contention with the just scheduled node, add it as a child of the scheduled node, and updated its ready time as the maximum among its previous ready time and the scheduled node's finish time; ▷ Lines 7 and 8 update $G^*$ after scheduling a node
9:      $Unscheduled = Unscheduled - 1$;
10: $L$ = maximal finish time of all the nodes;
11: return $\mathcal{E}$ and $L$;
```

5.2. Minimizing schedule length given a voltage and frequency assignment

The Earliest Task First (ETF) scheduling is a classic heuristic algorithm that tries to minimize a DAG's schedule length [24]. A task's ready time can be calculated as the maximum finish time of all its parents. The ETF heuristic always schedules the task with the earliest ready time first, when the corresponding resources become available. We also adopt the ETF strategy to schedule the transformed DAG. As has been mentioned, in our problem, if two communications (with the same ready time) need to use the same link(s), they cannot occur at the same time. In other words, they must be serialized; one has to be executed before the other. Thus, the question arises: which communication should execute first in order to minimize the schedule length?

Before describing our strategy, we borrow some concepts from [24]. A node with no parent is called an *entry* node, and a node with no child is called an *exit* node. The *b-level* of a node is the longest time between the start time of itself and the finish time of an exit node. The b-level of a node can be calculated recursively. First, for an exit node, the b-level is just the node's own execution time. For any other node, if all of its children's b-levels have been calculated, its b-level is just the sum of the maximal b-level of its children and its execution time.

When there might be a communication contention, our scheduling policy is to schedule the communication node that has the Greatest b-level First (GBF). Other unscheduled communication nodes that have potential contention with this event (i.e., use some same link(s)) will be considered a child of this node. Notice that, when XY-routing is chosen and a mapping is given, whether two communications have potential contentions can be determined according to the coordinates of their source tasks and destination tasks. Algorithm 2 gives the details of our overall scheduling scheme.

Fig. 3 is an illustration example of the scheduling algorithm, given a voltage and frequency assignment. Notice that, given a voltage and frequency assignment, all the execution times of tasks and communications can been calculated. All values in Fig. 3

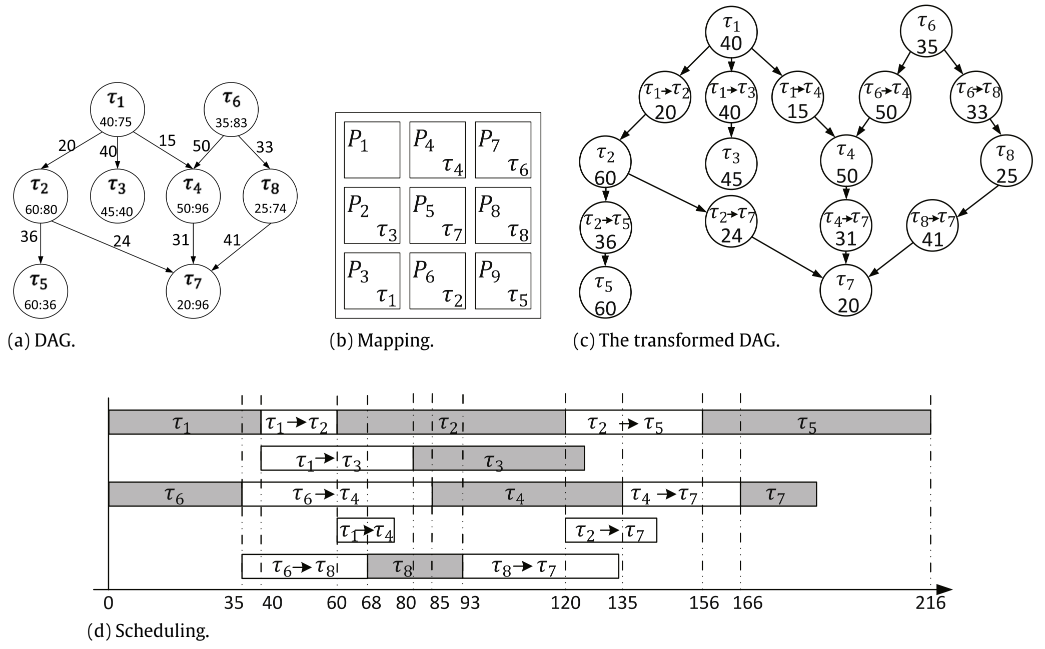

Fig. 3. An illustrative example. (a) The original DAG with task execution times and communication times calculated; each circle represents a task; the first number in the circle is the execution time requirement, and the second number in the circle is the total amount of in and out traffic with other tasks. (b) A mapping derived by the iterative rounding algorithm. (c) The transformed DAG, in which each node can represent either a task or a communication. (d) The scheduling derived by Algorithm 2.

are calculated assuming processors and NoC links operate at the highest available values. At the beginning, τ_1 and τ_6 are ready at the same time 0; τ_1 has the greatest b-level; so, τ_1 can be scheduled first. After that, τ_6 will become the one with the earliest ready time and be scheduled. Then, communications $\tau_1 \to \tau_2, \tau_1 \to \tau_3$, $\tau_1 \to \tau_3, \tau_6 \to \tau_4$, and $\tau_6 \to \tau_8$ become ready. Communications $\tau_1 \to \tau_2$ and $\tau_1 \to \tau_4$ have contention because they both need to use the link from processor P_3 to processor P_6. Since $\tau_1 \to \tau_2$ has a greater b-level than $\tau_1 \to \tau_4, \tau_1 \to \tau_2$ will be scheduled first; then, $\tau_1 \to \tau_4$ will add $\tau_1 \to \tau_2$ as a father, and $\tau_1 \to \tau_4$'s ready time will be updated as $\tau_1 \to \tau_2$'s finish time. The overall scheduling with a schedule length of 216 is shown in Fig. 3(d).

5.3. Genetic algorithm for voltage and frequency assignment

We have solved the problem of minimizing schedule length given a mapping and a voltage and frequency assignment for both tasks and communications. Next, we consider how to determine the optimal feasible voltage and frequency assignment. We develop a Genetic Algorithm (GA) to solve the problem. GAs search for good solutions to a problem among a large number of possible solutions, called a population. GAs begin with a set of candidate solutions. A new population is created from solutions of an old population in the hope of getting a better population. Solutions which are chosen to form new solutions (the next generation) are selected according to their fitness values. The more fitter the solutions are, the bigger chances they have to reproduce. This process is repeated until the number of generations reaches a predefined *Limit* or when the population converges.

In our problem, a solution can be represented by a vector with length $|T| + |E|$, denoted by

$$s = (s_1, s_2, \ldots, s_{|T|}, s_{|T|+1}, s_{|T|+2}, \ldots, s_{|T|+|E|}),$$

where $s_i \in v, (1 \le i \le |T|)$, is the voltage level for task τ_i, and $s_{|T|+i} \in f, (1 \le i \le |E|)$ is the frequency level for communication e_i. For a solution s, we denote $\mathcal{E}_s$ and L_s as the energy consumption and schedule length derived by Algorithm 2, respectively. We define the fitness value of a solution as follows:

$$fitness_s = \begin{cases} \dfrac{1}{\mathcal{E}_s} & \text{if } L_s \le Deadline; \\ \dfrac{\frac{1}{\mathcal{E}_s}}{10\left(\frac{L_s}{Deadline}\right)^2} & \text{if } L_s > Deadline. \end{cases} \quad (19)$$

Since our goal is to minimize the energy consumption, the solution with less energy consumption will be fitter than the one with greater energy consumption. However, if a solution's schedule length exceeds the deadline of the application, it is actually infeasible. Thus, its fitness value should be reduced significantly, but not to zero. In practical systems, power consumption may be proportional to the cube of operating frequency. This fact may result in an approximate relation between $\mathcal{E}_s$ and L_s: $\mathcal{E}_s \propto 1/L_s^2$. When $L_s > Deadline$, dividing $1/\mathcal{E}_s$ by $(L_s/Deadline)^2$ can avoid the situation where a very large schedule length still makes the scheduling have a great fitness value. The definition in Eq. (19) reflects these considerations.

We adopt the roulette-wheel selection algorithm, in which the possibility of a solution being selected as parents for the next generation is proportional to the fitness value of the solution. We apply the single point crossover for two parents to produce two descendants. For example, for two solutions,

$$s^1 = (s_1^1, s_2^1, \ldots, s_{|T|+|E|}^1) \quad \text{and}$$
$$s^2 = (s_1^2, s_2^2, \ldots, s_{|T|+|E|}^2),$$

we randomly generate the locus, l, for crossover. Then, the two descendants will be

$$s^{1'} = (s_1^1, s_2^1, \ldots, s_l^1, s_{l+1}^2, \ldots, s_{|T|+|E|}^2) \quad \text{and}$$
$$s^{2'} = (s_1^2, s_2^2, \ldots, s_l^2, s_{l+1}^1, \ldots, s_{|T|+|E|}^1).$$

For a mutation process, we denote the mutation probability of each element of the solution by p_m. For s_i, $(1 \le i \le |T|)$, the probability of it mutating to another voltage v_k, $(v_k \ne s_i)$ is $p_m/(n_V - 1)$;

Algorithm 3 GAVFA: Genetic Algorithm for Voltage and Frequency Assignment

```
1: Randomly generate a population of solutions;
2: while the total number of generations is less than or equal to Limit and the population does not converge do
3:     for each solution $s$ in the population do
4:         $(\mathcal{E}_s, L_s)$ =ETFGBF$(T, E, s)$;
5:         Calculate $fitness_s$ according to Equation (19);
6:     Find the two feasible solutions that have the greatest fitness values as two solutions in the next generation;
7:     Select the solutions as the parents for the next generation according to the Roulette-wheel algorithm;
8:     Conduct single point crossover to generate the next generation;
9:     Conduct mutation on each solution;
10: return the feasible solution with the greatest fitness value and the corresponding scheduling;
```

for $s_{|T|+i}$, $(1 \leq i \leq |E|)$, the probability of it mutating to another frequency f_k, $(f_k \neq s_i)$ is $p_m/(n_F - 1)$. For producing the next generation, we apply the elitist scheme [25], which keeps the two feasible solutions that have the greatest fitness values as two solutions in the next generation. Since, in each generation, two best feasible solutions are kept, the final solution quality and schedulability are expected to be improved. The overall approaches and procedures are described in Algorithm 3.

6. Simulations

In this section, we conduct extensive simulations to verify our model and overall method. Since our overall method first uses the iterative scheme to derive a mapping, and then assumes variable voltages and variable frequencies, we denote it as *it_vv_vf*, representing *it*erative mapping with *v*ariable *v*oltages and *v*ariable *f* requencies. We compare our model/method with five other models and/or methods.

Light-Weight Mapping with Variable Voltages and Variable Frequencies (lw_vv_vf): this method applies a light-weight algorithm to generate a task-to-processor mapping. Based on this mapping, it also applies the genetic algorithm, Algorithm GAVFA combined with Algorithm ETFGBF, to derive the final scheduling, including the voltage and frequency assignment. We notice that Algorithm RIRAM is quite time consuming to generate a mapping, due to iterative solving quadratic programming problems, especially when the numbers of processors and tasks are large. Thus, we come up with a light-weight algorithm that can derive a mapping efficiently without sacrificing too much performances. We briefly describe this light-weight algorithm as follows. First, we determine the minimal target square in the NoC that can hold all the application tasks. Then, given the original application DAG and the selected target square, we assign all the entry nodes of the graph to the processors close to the top-left corner of the target square, and assign all the exit nodes to the processors close to the bottom-right corner of the target square. After this we do a breadth-first traversal from the entry nodes, and a breadth-first traversal from the exit nodes by turns to assign the rest nodes of the graph; when assigning each node, we choose the best location for it such that it has the minimum weighted communication distances with nodes that have already been assigned. The algorithm terminates when we finished assigning all nodes to processors.

Random Mapping with Variable Voltages and Variable Frequencies (rd_vv_vf): this method randomly generates a valid task-to-processor mapping; then, it also applies the genetic algorithm, Algorithm GAVFA combined with Algorithm ETFGBF, to derive the final scheduling, including the voltage and frequency assignment.

Table 1
Processor configurations.

Voltage, v_j (V)	0.75	1.0	1.3	1.6	1.8
Frequency, f_j^v (MHz)	150	400	600	800	1000
Power, p_j^v (mW)	80	170	400	900	1600

Table 2
NoC link configurations.

Frequency, f_j (MHz)	200	400	600	800	1000
Bandwidth, B_j^f (Gbps)	6.4	12.8	19.2	25.6	32
Power consumption, p_j^f (mW)	160	180	520	880	1600

Iterative Mapping with Variable Voltages and Fixed Frequencies (it_vv_ff): this method also applies the iterative scheme to derive a mapping; then it assumes that the links' frequencies are fixed at the highest value and cannot be changed. However, the processors' operating voltages are variable. The genetic algorithm is also used. In this method, it only needs to determine the voltage levels for tasks or the corresponding processors.

Iterative Mapping with Fixed Voltages and Variable Frequencies (it_fv_vf): this method is similar to *it_vv_ff*; the difference is that it assumes that the processors' voltages are fixed, and that links' frequencies are variable.

Iterative Mapping with Fixed Voltages and Fixed Frequencies (it_fv_ff): this method is similar to both *it_vv_ff* and *it_fv_vf*; the difference is that it assumes that the processors' voltages and links' frequencies are all fixed. Thus, the genetic algorithm is not needed. Algorithm ETFGBF is applied directly to derive a scheduling.

6.1. Simulation settings

We apply our methods to several randomly generated graphs, as well as graphs from a real application. The tasks' execution requirements and communication volumes are generated such that the time for computation and the time for communication are comparable; in other words, neither dominates the other, and neither of them can be omitted. In this situation, the application slack can be utilized by either computations or communications in order to save the overall system energy consumption. For the processors, we adopt the Intel XScale processor's power configurations as shown in Table 1 [13,16]. The configuration for NoC links are provided in Table 2. NoC links' bit width is $b_w = 32$. These settings reflect practical configurations [23]. Application tasks' execution requirements are randomly generated within $[10, 100] \times 10^6$, in the unit of processor's clock cycles. Communication volumes are randomly generated within $[80, 800] \times 10^6$, in the unit of bits. E_{Rbit} is set as 0.01 nJ, such that the power consumption on links is comparable to that of on routers. The deadline of an application is set as twice that of the *it_fv_ff* such that we have a reasonable slack for energy reduction.

6.2. Simulations with randomly generated graphs

We randomly generate several application graphs, and run our algorithms on these graphs. The application characteristics and experimental results are provided in Table 3. The first column is the application number. The first, second, third, and fourth columns under the "application characteristics" are the number of tasks in the application, the number of communications in the application, the application tasks' total execution requirement, and the application's total communication volume, respectively. Due to space limit, we do not draw the application graphs. The "experimental results" include six fields, namely, TE0, TE1, TE2, TE3, TE4, and TE5, representing the **t**otal **e**nergy consumption

Table 3
Simulation applications and results.

No.	NoC dim	Application characteristics				Simulation results					
		$\|T\|$	$\|E\|$	cp/10^6	cm/10^6	TE0	TE1	TE2	TE3	TE4	TE5
1	3 × 3	7	5	350	2 064	269.94	273.53	288.56	317.40	627.08	704.48
2	3 × 3	7	9	324	4 584	348.21	393.46	431.80	512.68	682.23	878.88
3	3 × 3	8	7	365	2 760	327.22	324.83	358.43	396.46	684.50	806.00
4	3 × 3	8	13	402	7 176	527.51	627.56	670.88	817.81	922.05	1267.20
5	4 × 4	12	11	582	6 900	576.64	696.71	739.14	866.85	1197.40	1524.40
6	4 × 4	12	15	675	8 208	724.65	786.90	912.12	1069.70	1416.70	1838.20
7	4 × 4	13	12	689	6 432	710.56	700.03	821.10	887.24	1356.90	1671.40
8	4 × 4	14	18	747	9 852	902.32	931.77	1170.37	1279.96	1608.84	2127.48
9	6 × 6	28	33	1385	14 650	1854.86	1789.63	2524.86	2688.94	3102.40	4192.50

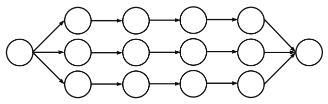
Fig. 4. ATR graph.

of *it_vv_vf*, *lw_vv_vf*, *rd_vv_vf*, *it_vv_ff*, *it_fv_vf*, and *it_fv_ff*, respectively. All energy consumption values are in the unit of nJ.

As we can see, the proposed *it_vv_vf* model and algorithms achieve a much less energy consumption than all others, except the *lw_vv_ff* algorithm. Taking the No. 3 application as an example, the energy reductions of *it_vv_vf* compared to *rd_vv_vf*, *it_vv_ff*, *it_fv_vf* and *it_fv_ff* are 8.7%, 17.5%, 52.2%, and 59.4%, respectively. Since processors consume most of the power, up to 90% in our experiments for different models, fixing the voltage levels of processors significantly limits the platform's ability to utilize application's slack to reduce energy consumption; thus, the energy consumptions of *it_fv_vf* and *it_fv_ff* are much higher than others. Allowing the processors to operate on multiple voltage levels, the energy consumption on processors is significantly reduced; this also results in a significant reduction in the total system energy consumption. By allowing NoC links to operate on variable frequencies, the platform can better utilize the application's slack to reduce total energy consumption; as a result, the total energy consumption of *it_vv_vf* is always less than that of *it_vv_ff*. Comparing *it_vv_vf* and *lw_vv_vf* with *rd_vv_vf*, we can see that the relaxation-based iterative rounding algorithm and the light-weight algorithm help to derive a better task-to-processor mapping.

The proposed light-weight method (*lw_vv_vf*) reduces the simulation time significantly. At the same time, it also achieves comparable energy consumption values as those of the *it_vv_vf* method. For most of the application graphs, *lw_vv_vf* achieves greater energy consumption values than *it_vv_vf*. For applications 3, 7, and 9, the energy consumption achieved by *lw_vv_vf* is even slightly less than that of *it_vv_vf*. We can see that, though *lw_vv_vf* is much efficient than *it_vv_vf*, it is not a very stable method. For example, it results in 21.0% more energy consumption compared to *it_vv_vf* in application 5.

6.3. Simulations with real application graphs

We also conduct simulations on real-world application graphs obtained from the Automatic Target Recognition (ATR) application, which does pattern matching of targets in images [26]. ATR graphs are different for different numbers of target detections in an image. We choose the one corresponding to 3 target detections; the graph is shown in Fig. 4. The application is mapped to a 4-by-4 NoC-based MPSoC. We conduct experiments on two cases of the application graph.

In the first case, the total task execution requirement of the application is 700 × 10^6, and the total communication volume is

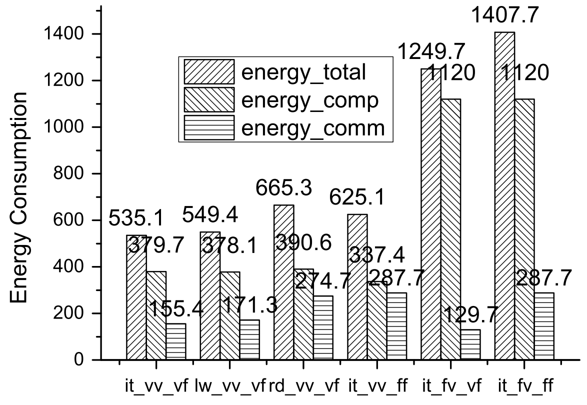

Fig. 5. Results of the ATR application (Case 1).

3390 × 10^6. Fig. 5 shows related energy consumption values of this case. "energy_total" represents the overall energy consumption of processors, NoC links and routers; "energy_comp" represents the energy consumption for computation, i.e., the energy consumption on processors; "energy_comm" represents the energy consumption for communication, i.e., the energy consumption on NoC links and routers. All energy consumption values are in the unit of nJ. As we can see, our proposed model and developed algorithm, *it_vv_vf*, achieves the lowest overall energy consumption. In *it_fv_ff*, all processors can only operate at the highest voltage level, and all links can only operate at the highest frequency level; thus, *it_fv_ff* cannot utilize the application slack for energy reduction, and its energy consumption is much greater than that of others. When the links' frequencies are fixed, only voltage scaling can utilize the slack for energy reduction; compared to *it_vv_vf*, the processors' energy consumption of *it_vv_ff* is even less, since voltage scaling can utilize all the slack for energy reduction on processors; however, the overall energy consumption of *it_vv_ff* is still greater than that of *it_vv_vf*, because *it_vv_ff* prevents the use of frequency tuning to better utilize the application slack. Compared with *it_vv_vf*, link frequency tuning of *it_fv_vf* can utilize all the slack for energy reduction on links; thus, the energy consumption for communication is less than that of *it_vv_vf*; however, the overall energy consumption of *it_fv_vf* is still greater than that of *it_vv_vf*, because *it_fv_vf* prevents the use of voltage scaling on processors to better utilize the application slack. In sum, the proposed model, *it_vv_vf*, which allows both processors and NoC links to utilize the application slack in a coordinated and complementary way, can achieve the greatest overall energy reduction.

In the second case, we double the volume of each communication in the first case, while any other aspects are kept unchanged. Fig. 6 shows related energy consumption values of this case. We can see that, the energy consumption for communications of *rd_vv_vf*, and *it_vv_ff* even exceeds the energy consumption for

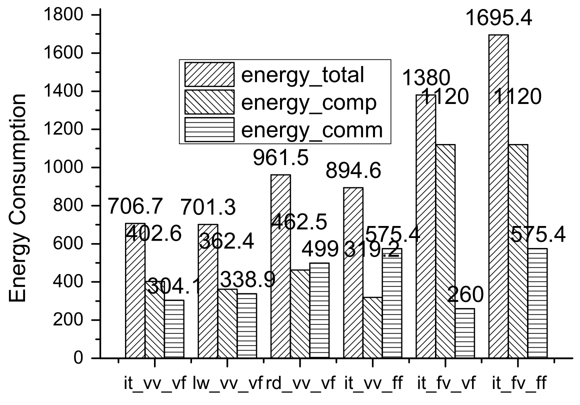

Fig. 6. Results of the ATR application (Case 2).

computation. In this situation, allowing links to adjust their frequency can save more energy. For example, in Fig. 5 the energy saving of *it_vv_vf* over *it_vv_ff* is 19.6%; in Fig. 6 the energy saving of *it_vv_vf* over *it_vv_ff* increases to 26.5%.

In both of the two cases, compared to *it_vv_ff*, the light-weight *lw_vv_ff* method reduces the simulation time significantly, and results in similarly low total energy consumption values.

7. Conclusion and future work

In this paper, we address the energy-efficient contention-aware application mapping and scheduling problem on NoC-based MPSoCs. We present a model where processors' voltage scaling and NoC links' frequency tuning can be combined together to reduce the overall system energy consumption. A two-step approach is adopted. First, the application mapping problem, which aims to find the mapping that minimizes the communication energy, is formulated as a quadratic integer programming problem, and solved by a relaxation-based iterative rounding scheme; we also provide a light-weight algorithm as an alternative, which reduces the overall algorithm complexity significantly and achieves comparable energy saving levels. The second problem, the application scheduling problem, is solved by a developed genetic algorithm, combined with a scheduling algorithm based on the ETF strategy. Finally, a mapping and scheduling for the application is derived, which significantly reduces the overall system energy consumption, verifying that jointly utilizing dynamic voltage scaling on processors and frequency tuning on NoC links provides great potential for overall energy reduction in MPSoCs.

Our work in this paper is for offline/static settings, where we know the application characteristics before-hand, and can batch consider them together. Practical systems may involve on-line application settings with or without our required characteristics. If the incoming application have known characteristics as in our model, i.e., the computation requirement and communication requirement, we can still formulate the incremental scheduling problem in a similar way as in our current problem formulation. The incremental scheduling will be much simpler and easier to solve. But, we may arrive at good local solution, but may deteriorate the overall system performance in a long run. Discussions on the general online versions of the problem is beyond the scope of this paper, and are left for future work.

References

[1] C.S. Behere, S. Gugulothu, Power reduction in network on chip links, in: Green Computing Communication and Electrical Engineering, ICGCCEE, International Conference on, March 2014, pp. 1–4.

[2] L. Benini, G. De Micheli, Networks on chips: a new SoC paradigm, Computer 35 (1) (2002) 70–78.

[3] S. Chai, Y. Li, J. Wang, C. Wu, An energy-efficient scheduling algorithm for computation-intensive tasks on NoC-based MPSoCs, J. Comput. Inf. Syst. 9 (5) (2013).

[4] J.-J. Chen, C.-F. Kuo, Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms, in: Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2007, pp. 28–38.

[5] C.-L. Chou, R. Marculescu, Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels, in: Proc. 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, 2007, pp. 161–166.

[6] C.-L. Chou, R. Marculescu, Contention-aware application mapping for network-on-chip communication architectures, in: Proc. IEEE International Conference on Computer Design, 2008, pp. 164–169.

[7] C.-L. Chou, R. Marculescu, User-aware dynamic task allocation in Networks-on-Chip, in: Proc. Design, Automation and Test in Europe, 2008, pp. 1232–1237.

[8] W.J. Dally, B. Towles, Route packets, not wires: On-chip inteconnection networks, in: Proc. 38th Annual Design Automation Conference, 2001, pp. 684–689.

[9] M. Gaur, V. Laxmi, M. Zwolinski, M. Kumar, N. Gupta, Ashish, Network-on-chip: Current issues and challenges, in: VLSI Design and Test, VDAT, 19th International Symposium on, 2015, pp. 1–3.

[10] P. Ghosh, A. Sen, A. Hall, Energy efficient application mapping to NoC processing elements operating at multiple voltage levels, in: Proc. 3rd ACM/IEEE International Symposium on Networks-on-Chip, 2009, pp. 80–85.

[11] P. Ghosh, A. Sen, A. Hall, Energy efficient application mapping to NoC processing elements operating at multiple voltage levels, in: Proc. 3rd ACM/IEEE International Symposium on Networks-on-Chip, 2009, pp. 80–85.

[12] J. Huang, C. Buckl, A. Raabe, A. Knoll, Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems, in: Proc. 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 2011, pp. 447–454.

[13] Intel XScale microarchitecture. http://developer.intel.com/design/intelxscale/benchmarks.htm.

[14] Y.-H. Kao, H. Chao, Design of a bufferless photonic Clos network-on-chip architecture, IEEE Trans. Comput. 63 (3) (2014) 764–776.

[15] K. Katayama, H. Narihisa, Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem, European J. Oper. Res. 134 (1) (2001) 103–119.

[16] W.Y. Lee, Energy-saving DVFS scheduling of multiple periodic real-time tasks on multi-core processors, in: Proc. 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, 2009, pp. 216–223.

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

[18] D. Li, J. Wu, Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms, in: Proc. 41st International Conference on Parallel Processing, 2012, pp. 430–439.

[19] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, F. Hessel, Exploring NoC mapping strategies: An energy and timing aware technique, in: Proc. Conference on Design, Automation and Test in Europe, 2005, pp. 502–507.

[20] A.K. Mishra, O. Mutlu, C.R. Das, A heterogeneous multiple network-on-chip design: An application-aware approach, in: Proceedings of the 50th Annual Design Automation Conference, 2013, pp. 36:1–36:10.

[21] S. Mukherjee, P. Bannon, S. Lang, A. Spink, D. Webb, The Alpha 21364 network architecture, IEEE Micro 22 (1) (2002) 26–35.

[22] P.K. Sahu, S. Chattopadhyay, A survey on application mapping strategies for network-on-chip design, J. Syst. Archit. 59 (1) (2013).

[23] L. Shang, L.S. Peh, N.K. Jha, Dynamic voltage scaling with links for power optimization of interconnection networks, in: Proc. Ninth International Symposium on High-Performance Computer Architecture, 2003, pp. 91–102.

[24] M. Shang, S. Sun, Q. Wang, An efficient parallel scheduling algorithm of dependent task graphs, in: Proc. of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2003, pp. 595–598.

[25] D. Thierens, Selection schemes, elitist recombination, and selection intensity, in: Proc. 7th International Conference on Genetic Algorithms, Morgan Kaufmann, 1998, pp. 152–159.

[26] R. Xu, R. Melhem, D. Mosse, Energy-aware scheduling for streaming applications on chip multiprocessors, in: 28th IEEE International Real-Time Systems Symposium, December 2007, pp. 25–38.

[27] T.T. Ye, L. Benini, G. De Micheli, Analysis of power consumption on switch fabrics in network routers, in: Proc. 39th Design Automation Conference, 2002, pp. 524–529.

[28] J. Zhan, J. Ouyang, F. Ge, J. Zhao, Y. Xie, DimNoC: A dim silicon approach towards power-efficient on-chip network, in: Design Automation Conference, DAC, 52nd ACM/EDAC/IEEE, 2015, pp. 1–6.

[29] J. Zhan, N. Stoimenov, J. Ouyang, L. Thiele, V. Narayanan, Y. Xie, Designing energy-efficient NoC for real-time embedded systems through slack optimization, in: Proc. 50th Annual Design Automation Conference, 2013, pp. 37:1–37:6.

[30] W. Zhang, L. Hou, J. Wang, S. Geng, W. Wu, Comparison research between xy and odd–even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip, in: Proc. WRI Global Congress on Intelligent Systems—Vol. 03, 2009, pp. 329–333.

[31] W. Zhou, Y. Zhang, Z. Mao, An application specific NoC mapping for optimized delay, in: Proc. International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006, pp. 184–188.

Dawei Li is a Ph.D. candidate in the Department of Computer and Information Sciences at Temple University since September 2011. He earned the Bachelor's degree from the Advanced Class, Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, Hubei, People's Republic of China. His research interest includes energy-aware task scheduling on multi-cores/multiprocessors, network-on-chip and data center networks.

Jie Wu is the chair and a Laura H. Carnell professor in the Department of Computer and Information Sciences at Temple University. He is also an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing and the Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, and ACM MobiHoc 2014, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.