UNISM: Unified Scheduling and Mapping for General Networks on Chip

Ou He, Sheqin Dong, Wooyoung Jang, *Student Member, IEEE*, Jinian Bian, *Member, IEEE*, and David Z. Pan, *Senior Member, IEEE*

***Abstract*—Task scheduling and core mapping have a significant impact on the overall performance of network on chip (NOC). In this paper, a unified task scheduling and core mapping algorithm called UNISM is proposed for different NOC architectures including regular mesh, irregular mesh and custom networks. First, a unified model combining scheduling and mapping is introduced using mixed integer linear programming (MILP). Then, a novel graph model is proposed to consider the network irregularity and estimate communication energy and latency, since the number of network hops is not accurate enough for irregular mesh and custom networks. To make the MILP-based UNISM scalable, a heuristic is employed to speed up our method. Compared with two previous state-of-the-art works, experimental results show that more than 15% and 11.5% improvement on the execution time is achieved with similar energy consumption on average for regular mesh NOC. For irregular and custom NOC, the improvement is 27.3% and 14.5% on the execution time with 24.3% and 18.5% lower energy. Moreover, our method is scalable for large benchmarks in terms of runtime.**

***Index Terms*—Core mapping, network on chip (NOC), network topology, task scheduling.**

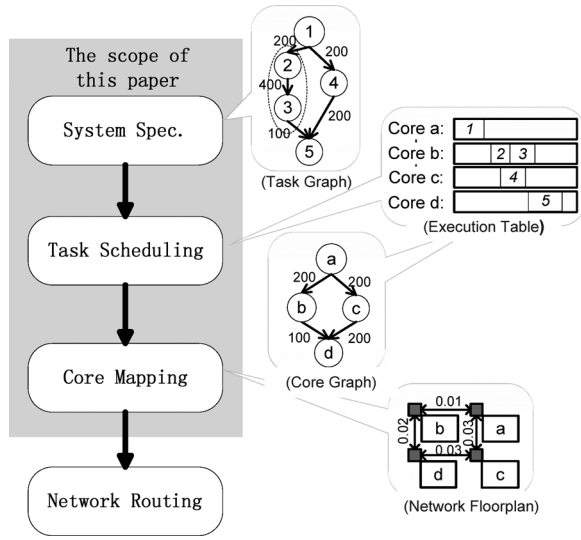


Fig. 1. Traditional NOC design flow.

I. Introduction

As VLSI technology advanced into deep submicrometer era, network on chip (NOC) which enhances on-die communication by data packetization is considered as an alternative of conventional bus-based interconnection in system-on-chip (SOC) design. As early stages of the NOC design flow, task scheduling and core mapping have a great impact on the overall performance of the entire system. Task scheduling is to assign each task in a task graph to different cores and decide the sequence of their executions (called execution table) if two tasks are scheduled on the same core. Then, a core graph, which defines the communication between cores instead of tasks, could be generated from the task graph. Core mapping is to assign cores in the core graph onto network tiles. Based on the floorplan which is generated after core mapping, network routing could be performed statically in design time or dynamically in run time.

In a traditional NOC design flow, task scheduling and core mapping are performed consequently, as shown in Fig. 1. The energy of network communications is not ignorable, compared with the energy spent on the computation of cores. Meanwhile, besides the time for running a task on a specified core, the communication latency also decides the total execution time of the input task graph. It means that inter-core communications also dominate the performance and energy of the entire design besides intra-core computations. However, communication can be optimized by both task scheduling and core mapping. As a result, it is necessary to consider scheduling and mapping as an integral process for a globally-optimized communication, which will further lead to a better performance and lower energy of the whole system.

Since scheduling and mapping are tightly coupled because of the communication, we will model them as a unified manner. Shin *et al.* [1] explored the design space of the unified methodology. Another two previous works were also carried out in a unified manner recently by Chi *et al.* [2] and Yu *et al.* [3]. However, only regular mesh was considered in these works. The benefit of this unified methodology has been shown by Ghosh *et al.* [4], but still for regular mesh.

Practically, there are different types of architectures for a NOC design, besides regular mesh. Fig. 2 shows three different NOC architectures. Numbers on the edges denote the relative

Manuscript received November 03, 2010; revised March 14, 2011; accepted May 10, 2011. Date of publication July 12, 2011; date of current version June 14, 2012. This paper was supported by the Ministry of Science and Technology of China International Cooperation Project (2011DFA60290).

O. He was with the Electrical and Computer Engineering Department, University of Texas, Austin, TX 78712 USA. He is now with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.

S. Dong and J. Bian are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: ho06@mails.tsinghua.edu.cn; dongsq@mail.tsinghua.edu.cn).

W. Jang and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2011.2159280

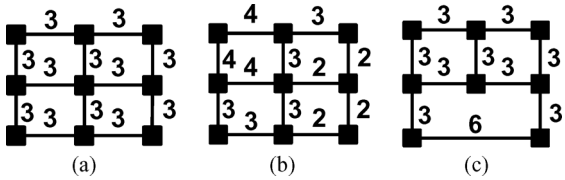


Fig. 2. Three architectures of NOC design. (a) Regular mesh. (b) Irregular mesh. (c) Custom NOC.

communication latency on each link. As mentioned before, this unified methodology has not been fully studied yet, especially for irregular mesh and custom network, as in Fig. 2(b) and (c). Besides, the benefit on chip performance and energy by applying this unified flow on irregular and custom network will be greater than regular mesh. On the other hand, it is difficult to model the network irregularity in these two architectures using a unified manner [5]. A detailed introduction about the benefits and challenges of a unified approach on different NOC architectures will be discussed in Section III.

In this paper, a unified task scheduling and core mapping named UNISM is proposed for all the architectures of NOC in Fig. 2. To the best of our knowledge, it is the first work to unify task scheduling and core mapping as an integral process for general NOC. Contributions are listed as follows.

1) A thorough study of the unified task scheduling and core mapping is presented using mixed integer linear programming (MILP).
2) A novel communication model called Labeled Graph is adopted to extend our MILP model to general networks, especially to irregular and custom networks, where the number of network hops is not accurate any more for the communication latency and energy.
3) A heuristic is developed to accelerate the UNISM algorithm and make it scalable for large benchmarks.

The rest of this paper is organized as follows. In Section II, previous work on scheduling and mapping are introduced. Then, in Section III the benefit and challenges by applying the unified methodology on architectures of NOC (i.e., regular, irregular and custom network) are illustrated. The problem is formulated in Section IV. Then, how to derive our unified MILP model and solve the network irregularity by Labeled Graph will be explained in Section V. After UNISM is developed, how to accelerate this model is discussed in Section VI. Experimental results are listed in Section VII followed by conclusions and future work in Section VIII.

II. Previous Works

Many previous works have been addressed on task scheduling and core mapping, targeting the design flow in Fig. 1.

For task scheduling, different algorithms have been adopted, such as integer linear programming (ILP) [6], Genetic Algorithm [7] and Ant Colony Algorithm [8]. Besides the system performance and energy consumption, deadline of the tasks is also considered in scheduling stage. Hu *et al.* [9] scheduled tasks on the custom NOC architecture considering real-time deadline and energy consumption. Chou *et al.* [10] learned user's behavior in their scheduling algorithm. Faruque *et al.* [11] proposed a dynamic task scheduling on distributed systems for

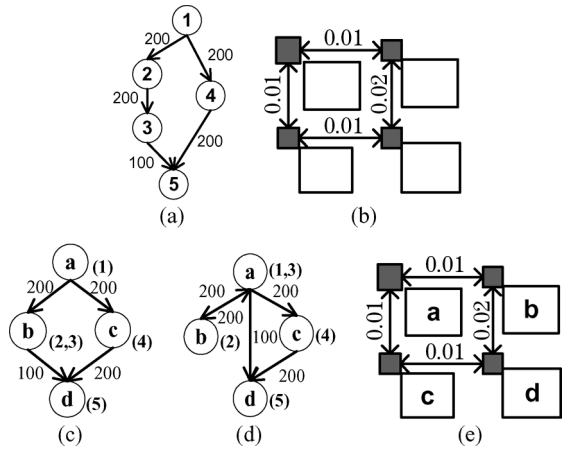


Fig. 3. Scheduling and mapping on irregular NOC. (a) Task graph. (b) Target networks tiles. (c) Core graph with min cut. (d) Core graph with larger cut. (e) Mapping result of Fig. 2(b).

NOC-based multiprocessor design. Kandemir *et al.* [12] put forward thread level scheduling in NOC rather than task level. Chen *et al.* [13] presented a compiler with the integration of task scheduling, core and data mapping and packet routing on regular mesh NOC. Ostler and Chatha [14] scheduled applications and data onto a specified architecture using ILP. Similarly, Kandemir *et al.* [15] also organized data and computation in a locality-aware manner to reduce communication energy. Tosun *et al.* [16] also adopted ILP to schedule applications onto cores. Only mesh structure was considered.

Core mapping has also been studied recently. A branch and bound method was presented in [17] on regular mesh. Murali *et al.* [18] implemented a fast mapping method called NMAP on mesh-based NOC with bandwidth constraints. For irregular mesh and custom NOC, core mapping is usually performed together with floorplanning information, like [19], [20]. These works take a core graph as an input, which describes the communications among different cores rather than tasks. As a result, a core graph needs to be extracted from the task graph before mapping. In order to produce a mapping-friendly core graph, some previous works performed min-cut partitioning on the task graph [19] [21]–[24]. With the sequential "min-cut partitioning + mapping" flow, Jang *et al.* [21] put forward the first work in core mapping called A3MAP for all three NOC architectures, as shown in Fig. 2.

III. Motivation

In this section, we illustrate the benefit of unified scheduling and mapping on irregular NOC. The challenge of supporting general network architectures will be discussed as well.

First, a task graph given in Fig. 3(a) is to be scheduled on *Core a, b, c,* and *d*. In some industrial benchmarks like *E3S* [25], the energy variance for the same task on different cores could be more than 10 times. Table I shows an example of the computing energy when tasks are executed on the four cores. The numbers in Table I are assumed for illustration. Then, a core graph could be generated as Fig. 3(c) or (d), where the nodes represent the cores and the edges represent the communication volume between two cores. At last, the core graph is mapped

TABLE I
TASK ENERGY ON EACH CORE

	Task 1	*Task 2*	*Task 3*	*Task 4*	*Task 5*
Core a	1	10	1	10	10
Core b	10	10	10	10	10
Core c	10	10	10	10	10
Core d	10	10	10	10	10

onto a given network in Fig. 3(b), where the weight on each edge denotes communication energy for sending one bit on this link.

Fig. 3(c) and (e) show scheduling and mapping results generated by a traditional "min-cut partitioning + mapping" flow, like [21]. In that case, the number of cut is 700 and the energy consumption is

$$E_{\text{core}} = 1 + 10 + 10 + 10 + 10 = 41$$
$$E_{\text{com}} = 0.01 * 200 * 3 + 0.02 * 100 = 8$$
$$E = E_{\text{core}} + E_{\text{com}} = 41 + 8 = 49$$

where E_{core} and E_{com} denote the core computing energy and communication energy, respectively.

However, there is another result from a unified flow with a larger cut (i.e., 900) in Fig. 3(d), which has a smaller total energy

$$E = E_{\text{core}} + E_{\text{com}}$$
$$= (1 * 2 + 10 * 3) + (0.02 * 800 + 0.02 * 100 * 2)$$
$$= 32 + 10 = 42.$$

A gap between local and global minimum of energy consumption is found, which originates from the irregularity of core computing and network communication. However, it is difficult to implement this unified flow on irregular mesh and custom network, because communication in the irregular networks is harder to model than regular mesh. The number of network hops (i.e., hop count), which is used to reflect the communication energy on regular mesh network, is no longer accurate. For example, in Fig. 3(e), the communication energy consumption from *Tile b* to *Tile d* is twice larger than the one from *Tile a* to *Tile b*. However, both of them have one hop (the same hop count). This problem was addressed as an open problem (P2) in [3]. This challenge will be further studied and solved in Section V-B.

IV. PROBLEM FORMULATION

The problem of unifying task scheduling and core mapping is formulated. Notations in this paper are defined in Table II.

Given

a) A directed acyclic task graph $G = (V, E)$, as shown in Fig. 3(a), where V is the set of tasks and E includes the edges which denote the communications between two tasks. Also, E is associated with $\{c_{ij}\}$, which means the set of communication volumes. Hard deadline constraints $\{d_i\}$ may be applied on several specified tasks as a subset of V.
b) A list of available cores $\{C_j\}$, which includes the execution time $\{w_{ij}\}$ for running *Task i* on *Core j* and the power of each core $\{p_j\}$, as in Table I.

TABLE II
DEFINITIONS OF NOTATIONS

x_{ij}	*Task i is scheduled on Core j (binary variables)*
y_{ij}	*Core i is mapped on Tile j (binary variables)*
z_{ij}	*Task i is scheduled on a core which is mapped on Tile j (binary variables)*
t_i	*the start time of Task i*
d_i	*hard deadline of Task i*
h_i	*the execution time of Task i*
w_{ij}	*the execution time for Task i on Core j*
c_{ij}	*communication volume from Task i to Task j*
l_{ij}	*communication latency from Task i to Task j*
e_{ij}	*communication energy from Task i to Task j*
p_j	*power of Core j*
E_{core}	*total computing energy on cores*
E_{com}	*total communication energy*
N	*the number of tasks in the task graph*
M	*the number of available cores*

c) A list of network tiles $\{T_k\}$ with routers and links has already been selected as well, as show in Fig. 3(b). The latency and energy consumption on each link are also given.

Find

a) A scheduling result x_{ij} from tasks to cores.
b) A mapping result y_{ij} from cores to network tiles.

Optimize

a) Total execution time (the time when finishing the whole task graph).
b) Energy consumption.
c) The total margin for the tasks finished before the deadline.

V. UNISM MODELING AND ALGORITHM

In this section, detailed techniques of our UNISM model will be discussed.

A. Unified Scheduling and Mapping Using MILP

1) Modeling the Objective Function: As mentioned in Section IV, the following objectives are optimized together as a tradeoff.

a) The execution time of the entire task graph: T.
b) Total energy consumption: $E = E_{\text{core}} + E_{\text{com}}$.
c) Total margin before task deadline: R.

The cost function (noted as C) is formulated as follows:

$$C = \alpha_0 T + \alpha_1 E + \alpha_2 R$$
$$= \alpha_0 T + \alpha_1 E_{\text{core}} + \alpha_1 E_{\text{com}} + \alpha_2 R$$
$$\text{where } \alpha_0, \alpha_1 \text{ and } \alpha_2 \text{ are weight factors.} \quad (1)$$

In (1), E_{core} is decided only by task scheduling, which will be defined in Section V-A2. The other objectives (T, E_{com}, and R) are relevant to both scheduling and mapping, which will be modeled in Section V-A4.

2) Modeling Task Scheduling: x_{ij} in Table II is the variable of task scheduling. In this section, the objectives and constraints which are only related to x_{ij} will be discussed.

First, we assume that one task cannot be scheduled on two cores at the same time. That means for each task $v_i \in V$

$$\sum_{j=0}^{M-1} x_{ij} = 1 \quad i = 0, 1, 2, \cdots, N-1. \tag{2}$$

When x_{ij} is ready, the objective E_{core} could be calculated

$$E_{\text{core}} = \sum_{i=0}^{N-1} \left(\sum_{j=0}^{M-1} x_{ij} w_{ij} p_j \right). \tag{3}$$

Meanwhile, there is another constraint, i.e., two tasks which are scheduled on the same core cannot be executed at the same time. This constraint is modeled by the following inequalities.

For each task $v_i, v_j \in V$

$$\sum_{k=0}^{M-1}(k*x_{ik}*A) - \sum_{k=0}^{M-1}(k*x_{jk}*A) + (t_i + h_i - t_j) \le A^2(1 - s_{ij})$$
$$\sum_{k=0}^{M-1}(k*x_{jk}*A) - \sum_{k=0}^{M-1}(k*x_{ik}*A) + (t_j + h_j - t_i) \le A^2 * s_{ij}$$

where s_{ij} is an auxiliary binary variable, k is the index for the traversal of all M cores, and A is a constant which is much greater than t_i, h_j, t_j, and h_j but less than A^2.

Next, further discussion will be given to explain how these two inequalities work.

If *Task i* and *Task j* are scheduled on the same core, then

$$x_{ik} \equiv x_{jk}, \quad k = 0, 1, \cdots, M-1.$$

We will know

$$\sum_{k=0}^{M-1}(k * x_{ik} * A) - \sum_{k=0}^{M-1}(k * x_{jk} * A) = 0.$$

These two inequalities can be simplified as

$$(t_i + h_i - t_j) \le A^2(1 - s_{ij})$$
$$(t_j + h_j - t_i) \le A^2 * s_{ij}$$

which means or $t_i + h_i \le +t_j$ or $t_j + h_j \le t_i$

If *Task i* and *Task j* are not scheduled on the same core, that means x_{ik} is not always equal to x_{jk}. As a result

$$\left| \sum_{k=0}^{M-1}(k * x_{ik} * A) - \sum_{k=0}^{M-1}(k * x_{jk} * A) \right| \ge A.$$

Because A is a constant which is much greater than t_i, h_i, t_j, and h_j, $(t_i + h_i - t_j)$, and $(t_j + h_j - t_i)$ are not very important in these inequalities. So, they could be simplified as

$$\sum_{k=0}^{M-1}(k * x_{ik} * A) - \sum_{k=0}^{M-1}(k * x_{jk} * A) \le A^2(1 - s_{ij})$$
$$\sum_{k=0}^{M-1}(k * x_{jk} * A) - \sum_{k=0}^{M-1}(k * x_{ik} * A) \le A^2 * s_{ij}$$

which could be always satisfied. So, there will be no restrictions for t_i, h_i, t_j, and h_j if *Task i* and *Task j* are not scheduled on the same core.

Moreover, h_i is the execution time for *Task i*, which is modeled using x_{ij} as follows:

$$h_i = \sum_{j=0}^{M-1}(x_{ij} * w_{ij}) \quad i = 0, 1, 2, \cdots, N-1. \tag{4}$$

3) Modeling Core Mapping: There are also several constraints in our MILP model, which are only relevant to the mapping variable y_{ij} It is known that one core cannot be mapped on two tiles in the network. That means $\exists c_i \in C$

$$\sum_{j=0}^{M-1} y_{ij} = 1 \quad i = 0, 1, 2, \cdots, M-1.$$

Meanwhile, one tile cannot be used for two cores. $\exists t_i \in T$

$$\sum_{i=0}^{M-1} y_{ij} = 1 \quad j = 0, 1, 2, \cdots, M-1. \tag{5}$$

4) Modeling the Unification of Scheduling and Mapping: As mentioned in Section V-A1, more objectives in (1) are related to both scheduling and mapping. Therefore, the unified modeling combining these two stages becomes the only way to evaluate these objectives.

The first challenge is how to design the variables, which can be used to calculate these objectives in a linear way. In this paper, z_{ij} is defined as unification variable, which equals *true* if and only if *Task i* is scheduled on a core that is mapped on *Tile j*. Therefore, z_{ij} decided by both x_{ij} and y_{ij} as in (6), which is not linear

$$z_{ij} = \sum_{k=0}^{M-1} x_{ik} y_{kj}. \tag{6}$$

Fortunately, since x_{ik}, y_{kj}, and z_{ij} are all binary variables, the quadratic relation between x_{ik} and y_{kj} in (6) can be substituted by two linear inequalities

$$\begin{cases} D * (1 - z_{ij}) \ge + \sum_{k=0}^{M-1} k(x_{ij} - y_{kj}) \\ D * (1 - z_{ij}) \ge - \sum_{k=0}^{M-1} k(x_{ij} - y_{kj}) \end{cases} \tag{7}$$

where D is a constant which satisfies $D \gg M$. Moreover, there is one constraint on z_{ij} in (8)*, which guarantees that Task i can only be scheduled and then mapped on one and only one network tile*

$$\sum_{j=0}^{M-1} z_{ij} = 1 \quad i = 0, 1, 2, \cdots, N-1. \tag{8}$$

Then, for binary variables x_{ik}, y_{kj} and z_{ij}, we will prove that the two inequalities in (7) will deduce the same value on z_{ij} as (6), as long as x_{ik} and y_{kj} are the same.

5) Proof: Because of (2) and (5), (6) has an equivalent form

$$z_{ij} = \begin{cases} 1, & \exists k \in [0, M-1], \text{s.t. } x_{ik} = 1 \text{ and } y_{kj} = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Then, we only need to prove (9) can be substituted by (7). From (7), we know that we get (10), shown at the bottom of the next page.

Since $D \gg M$, we can further simplify (10) as

$$z_{ij} = \begin{cases} 0 \text{ or } 1, & \text{if } \exists k \in [0, M-1], \text{ s.t. } x_{ik} = 1 \text{ and } y_{kj} = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

From (11), we first assume that

$$z_{ij} = \begin{cases} 0, & \text{if } \exists k \in [0, M-1], \text{ s.t. } x_{ik} = 1 \text{ and } y_{kj} = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

Considering (12), we will deduce that

$$\sum_{j=0}^{M-1} z_{ij} = 0 \quad (i = 0, 1, 2, \cdots, N-1). \tag{13}$$

Then, our assumption about (11) leads to a conflict between (8) and (13). So, in (11), we have

$$z_{ij} = \begin{cases} 1, & \exists k \in [0, M-1], \text{ s.t. } x_{ik} = 1 \text{ and } y_{kj} = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Equation (14) which derives from (7) is equal to (6). So, (7) can be a linear substitution of (6). ■

After the unification variable z_{ij} is ready, the communication latency l_{ij} and energy e_{ij} for sending one bit from *Task i* to *Task j* could be calculated by (15) and (16)

$$l_{ij} = f_l(z_{il}, z_{jm}) \tag{15}$$

$$e_{ij} = f_e(z_{il}, z_{jm}). \tag{16}$$

We only use f_l and f_e to denote this function. Further study will be discussed in Section V-B to derive the analytical form of f_l and f_e Because when applying the unification on all network architectures using f_l and f_e, there is another challenge: network irregularity.

Based on (15), we can formulate the data dependency constraint in (17). If there exists one communication from *Task i* to *Task j*, then

$$t_j \geq t_i + h_i + c_{ij} l_{ij}. \tag{17}$$

Since h_i has already been formulated in (4), two objectives T and R in (1) could be formulated as follows:

$$T = \max\{t_i + h_i\} \Leftrightarrow \begin{cases} \min\ T \\ \text{s.t. } T \geq t_i + h_i \end{cases} i = 0, 1, \cdots, N-1.$$

If *Task i* has a hard deadline d_i

$$\begin{cases} R = \sum_i (t_i + h_i - d_i) \\ t_i + h_i \leq d_i. \end{cases}$$

Given a network topology G={V, E}, V denotes all the network tiles and E denotes energy consumptions on each link
Find a set of Labeled Graphs {$L^{(k)}$}
Procedure
1) For any two nodes $v_i, v_j \in V$, we calculate the minimal energy between v_i and v_j (denoted as E_{Cij}) using *Floyd-Warshall* Algorithm. For example, the minimal energy from *Node b* to *Node f* in Fig. 5(a) is 4.
2) Choose one node v_i and label this node with 0.
3) Calculate the minimal energy (i.e. shortest path) from $v_i$to all the other nodes using *Dijkstra* Algorithm. Build a new Labeled Graph $L^{(i)}$, like Fig. 5(c) to 5(g). Label each node v_j with its minimal energy to v_i, which is denoted as $L_j^{(i)}$.
4) For any two nodes $v_j, v_k \in V$, if there is no node $v_i \in V$, *s.t.* $E_{Cjk} = \left(L_k^{(i)} - L_j^{(i)}\right)$, choose v_j and goto *Step 3*. For example, we find that the minimal energy from *Node b* to *Node f* can be calculated by a subtraction of the labels on the graph of Fig. 5(g), i.e. 4=4-0.
end Procedure

Fig. 4. Procedure of building labeled graphs.

Based on (16), the objective E_{com} can be calculated as

$$E_{\text{com}} = \sum_{(i,j) \in E} (c_{ij} e_{ij})$$

where E is the set of all communications in the task graph.

B. Extending UNISM to General Networks on Chip

Because of the network irregularity, f_l and f_e in (15) and (16) have not been solved yet in Section V-A4. In this section, we will focus on handling the network irregularity to solve (15) and (16). f_e will be studied in Section V-B1 as an example. f_l could be solved in the same way.

1) Labeled Graph: Fig. 5(a) shows an irregular mesh including six tiles. The number on the edge denotes the energy consumption on each link. Meanwhile, the energy of the router is also added into this number. Suppose E_{cmn} is the communication energy to transfer one bit from *Tile m* to *Tile n*. The most straightforward way to calculate the energy consumption for one communication is to enumerate all the pairs of tiles with their energy consumptions, as the following inequality:

$$e_{jk}^i \geq \sum_{m=0}^{5} \left(\sum_{n=0}^{5} \left(u_{mn}^i E_{Cmn} \right) \right)$$

where e_{jk}^i denotes the energy of Communication i, which starts from Task j to Task k. u_{mn}^i is a binary variable and equals true when and only when Task j is scheduled and mapped to Tile m and Task k is scheduled and mapped to Tile n.

New variables $\{u_{mn}^i\}$ are needed for the enumeration. The number of u_{mn}^i is $10 \times 6 \times 6 = 360$, suppose we have 10 communications and each one needs $6 \times 6 = 36$ variables to enumerate all the pairs of tiles. These variables may lower the

$$\begin{cases} D * (1 - z_{ij}) \geq 0, \\ D * (1 - z_{ij}) \geq 0 \left| \sum_{k=0}^{M-1} k(x_{ik} - y_{kj}) \right| > 0, \end{cases} \quad \begin{array}{l} \text{if } \exists k \in [0, M-1], \text{ s.t. } x_{ik} = 1 \text{ and } y_{kj} = 1 \\ \text{otherwise} \end{array} \tag{10}$$

TABLE III
MINIMAL ENERGIES FOR BUILDING LABELED GRAPHS

i -> j	E_{Cij}	$L_j^{(d)} - L_i^{(d)}$	$L_j^{(a)} - L_i^{(a)}$	$L_j^{(b)} - L_i^{(b)}$	max
a->b	**2**	1	**2**	-2	**2**
a->c	**1**	-1	**1**	0	**1**
a->d	**2**	-2	**2**	1	**2**
b->c	**2**	-2	-1	**2**	**2**
b->d	**3**	-3	0	**3**	**3**
c->d	**1**	-1	**1**	**1**	**1**
b->a	**2**	-1	-2	**2**	**2**
c->a	**1**	**1**	-1	0	**1**
d->a	**2**	**2**	-2	-1	**2**
c->b	**2**	**2**	1	-2	**2**
d->b	**3**	**3**	0	-3	**3**
d->c	**1**	**1**	-1	-1	**1**

efficiency of the MILP solver. A detailed comparison on MILP complexity is shown in Section V-B2.

In this paper, we explore the network irregularity and propose a graph model named Labeled Graph to calculate the energy with no additional variables. Taking the communication from *Tile b* to *Tile f* as an example, steps of building Labeled Graph are listed in Fig. 4.

Fig. 6 and Table III are adopted here for a better illustration on how to build Labeled Graphs. A network topology is given in Fig. 6(a). First, the minimal communication energy between each two tiles in Fig. 6(a) is calculated using Floyd-Warshall Algorithm. Results are shown in Column E_{Cij} of Table III.

Then, *Tile d* is picked up and the minimal energies from *Tile d* to all the other tiles are labeled in Fig. 6(b) as Labeled Graph $L^{(d)}$ using Dijkstra Algorithm. By subtracting the labels on $L^{(d)}$, the communication energy between each two tiles is calculated in the third column of Table III. Except those numbers printed in bold, we find that some other values in Column E_{Cij} cannot be represented by the numbers in the third column. So, *Tile a* is picked up and Labeled Graph $L^{(a)}$ is generated.

However, there are still some numbers which cannot be represented by the third and fourth columns of Table III. As a result, Graph $L^{(b)}$ is needed by picking *Tile b*. Then, all the numbers in Column E_{Cij} are represented at least once by Column 3, 4, and 5. From Theorem 1, we know that

$$E_{Cij} = \max\left\{L_j^{(d)} - L_i^{(d)}, L_j^{(a)} - L_i^{(a)}, L_j^{(b)} - L_i^{(b)}\right\}$$

as shown in the last column of Table III.

2) Theorem 1: Given a set of Labeled Graph $\{L^{(k)}\}$, the energy of the communication from *Tile i* to *Tile j* (termed as E_{Cij}) is

$$E_{Cij} = max\left\{L_j^{(k)} - L_i^{(k)}\right\}, \quad k = 0, 1, \cdots, K-1$$

where $L_i^{(k)}$ denotes the label value on Tile i in the k^{th} Labeled Graph and K is the total number of generated graphs.

a) Proof: First, in *Step b* and *c*, E_{Cij} can be hit at least once in $\{L^{(K')}\}$, i.e., $E_{Cij} = L_j^{(K')}$.

Second, for the other k's, we will prove

$$E_{Cij} \geq L_j^{(k)} - L_i^{(k)}, \quad k = 0, 1, \cdots, K'-1, K'+1, \cdots, K-1.$$

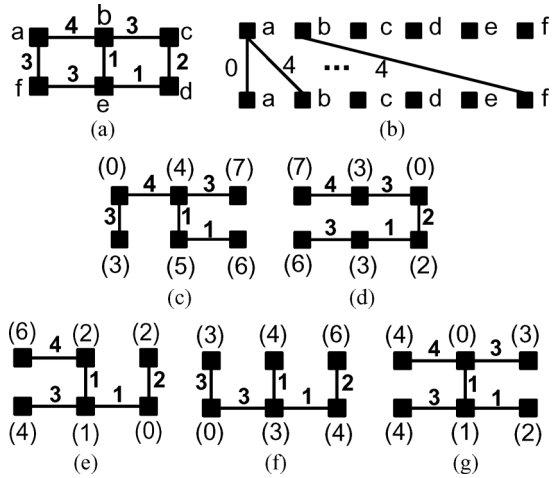


Fig. 5. Labeled graphs. (a) An irregular mesh. (b) Calculating minimal energy.

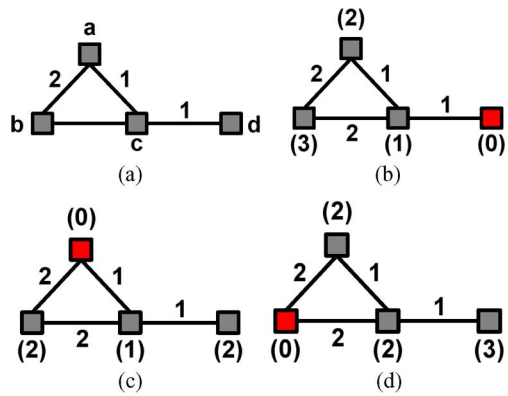


Fig. 6. Example of labeled graph.

According the symmetry of $\{L^{(k)}\}$ half of $L_j^{(k)} - L_i^{(k)}$ will be negative, which is obviously satisfied this inequality because $E_{\text{Cij}} > 0$.

For those $L_j^{(k)} - L_i^{(k)} > 0$ supposed for a Labeled Graph $L^{(K'')}$ we have

$$E_{Cij} < L_j^{(K'')} - L_i^{(K'')}.$$

Meanwhile, we assume $L^{(K'')}$ is generated from *Tile s* to all the other tiles in *Step b*. As shown in Fig. 7, from *Tile s* to *Tile j*, we have two paths: the minimal path (i.e., $s, \cdots, j$) and the path via *Tile i* (i.e., $s, \cdots, i, \cdots, j$). Moreover, the energy consumptions on those two paths are $(L_j^{(K'')} - L_s^{(K'')})$ and $(L_i^{(K'')} - L_s^{(K'')} + E_{Cij})$, respectively.

Because $E_{Cij} < L_j^{(K'')} - L_i^{(K'')}$ we can deduce that

$$\begin{aligned}&\left(L_i^{(K'')} - L_s^{K''} + E_{Cij}\right)\\ &< \left(L_i^{K''} - L_s(K'') + L_j(K'') - L_i(K'')\right)\\ &= \left(L_j^{''} - L_s(K'')\right).\end{aligned}$$

Since $(L_j^{(K'')} - L_s^{(K'')})$ is the path with minimal energy from *Tile s* to *Tile j*, it should be no greater than the energy of any other paths. However, it is greater than the energy on the path from *Tile s* to *Tile j* via *Tile i* (i.e., $s, \cdots, i, \cdots, j$). ■

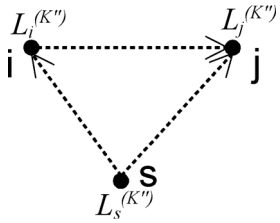


Fig. 7. Labeled Graph staring from *Tile s*.

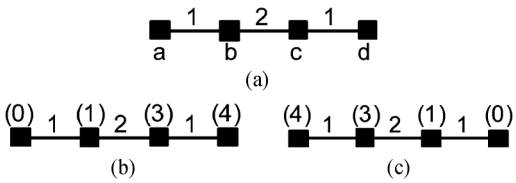


Fig. 8. Line-based topology and its labeled graphs.

When applying the procedure mentioned in Fig. 4, we do not need too many Labeled Graphs to satisfy *Step 4*. For regular mesh, $K = 4$(i.e., four corner tiles of the mesh). For the network in Fig. 5(a), $K = 5$ In fact, the order of picked tiles to generate a new graph will affect the number of required Labeled Graphs, even for the same network topology. In this paper, we use a heuristic to decide the order. Tiles with least degree (the number of related edges) will be picked first. After that, if the condition in *Step 4* is still not satisfied, we will pick those tiles which cannot be calculated by the existing graphs to generate a new graph. Generally, we will have the following Theorem.

3) Theorem 2: The total number of required Labeled Graphs (termed as K) is constrained by the following inequality:

$$2 \leq K \leq M.$$

a) Proof: To prove $2 \leq K$, we first give an example which only needs two Labeled Graphs. Then, we prove $K \neq 1$, since we already know $K \geq 1$.

Consider the line-based regular network topology, as in Fig. 8, we need two Label Graphs. One invokes *Dijkstra Algorithm* starting from *Tile a* while the other one is from *Tile d* to all the other tiles, as in Fig. 8(b) and (c).

Meanwhile, we cannot use just one Labeled Graph to satisfy the condition mentioned in *Step c*, which means $K \neq 1$. In *Step c*, the minimal energy between each two tiles should be calculated by subtracting the labels on these two tiles. Given two tiles i and j, the minimal energy from *Tile i* and *Tile j* (termed as E_{Cij}) could be calculated by

$$E_{Cij} = L_j^{(1)} - L_i^{(1)}.$$

However, we know that

$$E_{Cji} = L_i^{(1)} - L_j^{(1)}.$$

That means both E_{Cij} and E_{Cji} cannot be positive at the same time. Therefore, one Label Graph is always not enough, i.e., $K \neq 1$.

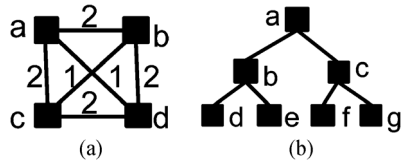


Fig. 9. Two network topologies. (a) Fully connected. (b) Tree structure.

On the other hand, we can start *Dijkstra Algorithm* from every tile to the other tiles. After that, we will get M Labeled Graphs. In this case, the minimal energy between each two tiles could be calculated anyway by one Label Graph at least. That means $K \leq M$. ■

Then, an example of "full connection" topology is given in Fig. 9(a), which needs M Labeled Graphs. This is the worst case in Label Graph. We also illustrate a "tree" topology and the number of required graphs is four, which equals the number of leaf nodes (i.e. *Tile d, e, f, and g*). Empirically for any randomly specified network topology, K is much less than M.

Based on *Theorem 1*, f_e in (16) can be formulated linearly

$$\begin{aligned} e_{ij} &= f_e(z_{il}, z_{jm}) \\ &= \max\left(\sum_{m=0}^{M-1}\left(z_{jm}L_m^{(k)}\right) - \sum_{l=0}^{M-1}\left(z_{il}L_l^{(k)}\right)\right) \\ & k = 0, 1, \cdots, K-1. \end{aligned} \tag{18}$$

Using Labeled Graph, the number of variables is 60 from (18), compared with 360 for the enumeration-based model.

4) Time Complexity: Then, time complexity of our MILP model is discussed in terms of the number of variables. Labeled Graph does not introduce any new variables. Thus, the number of variables used in MILP is $(3NM + M^2 + 2N + N_{ND})$, where N_{ND} is the number of task pairs which do not have data dependency in the task graph. Definitions of the other notations can be found in Table II.

In practical cases, N is usually greater than M (thus $NM \gg M^2$) and NM usually greater than N_{ND} Thus, the number of variables can be approximated as $O(NM)$, which has the same complexity as a single MILP-based scheduling problem. However, the aforementioned enumerative modeling will bring $O(NM^2)$ new variables, which makes the model harder to solve.

5) Modeling the Communication Latency: The communication latency can also be calculated effectively by assigning latency labels on the Labeled Graph. However, a precise estimation for the latency is harder than energy, since routing congestion may greatly change the latency.

It is hard to precisely estimate the congestion, since the routing stage has not been started yet in such early stages as task scheduling and core mapping. As a therapy, two heuristics are adopted to alleviate the impact caused by the routing congestion.

First, the communication with larger volume will tend to suffer from the congestion and has longer latency. Second, the path with more hops will be easily congested. For example, the path "a-b-e-d" in Fig. 5(a) has stronger possibility to be congested than "a-b", since more links are included in "a-b-c-d".

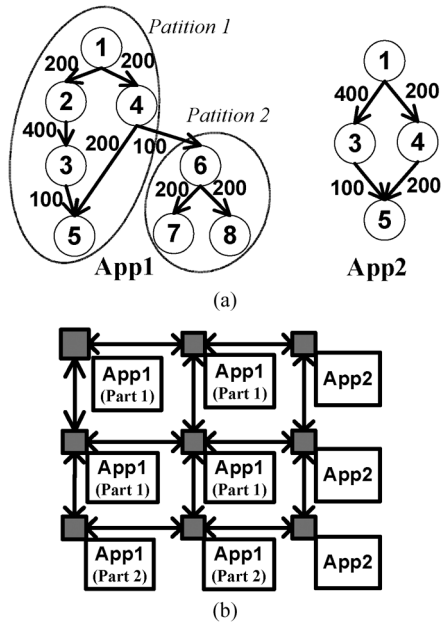


Fig. 10. Accelerating the optimization. (a) A task graph with two applications. (b) A localized unification on 3 × 3 mesh.

Given a list of sub task groups $\{G_i\}$, a list of unassigned cores $\{C_j\}$, a list of unassigned tiles $\{T_k\}$
Find a grouped cores and tiles for each sub task group of $\{G_i\}$
Procedure
5) Sort all the sub task groups in non-descending order by the number of tasks in G_i, which denotes as $|G_i|$. If two groups have the same number of tasks, the one with larger total runtime on each core (calculated by $\sum_{j=0}^{M-1}\left(\sum_{task\ v_i \in G_i} w_{ij}\right)$) will be sorted first.
6) Budget the number of cores, which will be assigned to each sub task group G_i. It is proportional to the number of tasks in G_i.
7) Build a bipartite graph $B = \left\{\{G_i\} \cup \{C_j\}, E_{ij}\right\}$, where E_{ij} denotes an edge between G_i and C_j and $E_{ij} = \left(\sum_{task\ v_i \in G_i} w_{ij}\right)/|G_i|$
8) Find the minimum weighted bipartite matching for Graph B
9) Sort all the sub task groups $\{G_i\}$ in non-descending order by the total communication volume within each group.
10) **for** each sub task group G_i
11) if G_i has communications with other assigned sub task groups
12) Pick up an unassigned tile T_k with the smallest link to these assigned groups
13) else
14) Pick up an unassigned tile T_k with smallest link
15) Find the closest $(N_i - 1)$ tiles to T_k using *Dijkstra* algorithm on the given network architecture, where N_i is the number of cores assigned to G_i
16) Assign these $(N_i - 1)$ tiles and T_k to G_i
end for
end Procedure

Fig. 11. Grouping cores and network tiles for sub task graphs.

In this paper, two penalty terms are added into the latency estimation. As a result, larger margin to the deadline is reserved to these easily congested communications

$$l'_{ij} = l_{ij} \cdot \left(1 + \beta \frac{c_{ij}}{c_{\mathrm{MAX}}}\right) \cdot \left(1 + \gamma \frac{n_{ij}}{n_{\mathrm{MAX}}}\right)$$

where l_{ij} is the latency from Task i to Task j calculated by Labeled Graph, and l'_{ij} is the latency after the penalty. c_{ij} and n_{ij} are communication volume and hop count while c_{MAX} and n_{MAX} are the largest volume and hop count of the entire design. β and γ are penalty percentages, which is empirically set to 20% in this paper.

We also find that the path with minimal energy could be different from the one with minimal latency sometimes. That means in this case we cannot route a path with both minimal energy and minimal latency at the same time. A tradeoff has to be made. In our method, we used a weighed objective function, see (1). For a tight deadline design, α_0 and α_2 could be relatively greater than α_1. For other designs which do not have tight deadlines, we can enlarge α_1 to save the energy. In this paper, the default value of α_1 is 0.5 and $\alpha_0 = 10\alpha_2$ That means α_0 equals 0.45 and α_2 equals 0.05.

VI. Speedup Techniques for UNISM

Since our UNISM model has been discussed in Section V, we will focus on how to accelerate this model in this section.

As pointed in [7], in practical designs, several independent applications (or called sub task graphs) will appear in the given task graph, as shown in Fig. 10(a). For a large sub task graph, such as *App 1* in Fig. 10(a), we can still partition it into several smaller subgraphs using min-cut partitioning.

In this situation, it is inefficient to merge all the sub graphs and perform an exhausting search on the solution space. A better tradeoff between solution quality and runtime is to localize each subgraph on the network, as shown in Fig. 10(b). All the cores and network tiles are partitioned into three groups and then each group performs scheduling and mapping within a local domain of the network. As discussed in Section III, min-cut partitioning may cause some loss on the solution quality. But different from the previous works which partition the task graph directly onto each core, we partition this graph onto different groups of cores, instead of one core. Within each group, we can still apply our UNISM. The granularity of our partitioning is much larger than the previous works which results in a better tradeoff between CPU runtime and solution quality, as in Section VII. Our method for accelerating UNISM is introduced as follows:

First, if the number of input applications (or called sub graphs) is greater than a threshold, they will be clustered to this threshold. On the other hand, if the number of the applications is less than the threshold, they will be partitioned to this threshold. In our implementation, the threshold for grouping sub task graphs is empirically set to $\max(3, [M/4])$. One clustered or partitioned group of applications is called a *sub graph group*.

Then, cores and network tiles are assigned to each sub graph group. The procedure of grouping cores and network tiles is listed in Fig. 11.

An example will be discussed to show how the pseudo code works in Fig. 11. First, a task graph of two applications is given, as in Fig. 12(a). Each task has a runtime w_{ij} on each core, as in Table IV. These numbers can be fixed when the task type and core type are fixed. They are usually defined in the benchmarks. In Table IV, they are assumed for illustration. Fig. 12(b) gives the network topology, where numbers on each link represent the weighted averages between latency and energy. For the illustration, the threshold is set to 3.

First, since the number sub task graphs is two which is less than the threshold, we partition App 1 into two parts, as in Fig. 12(a).

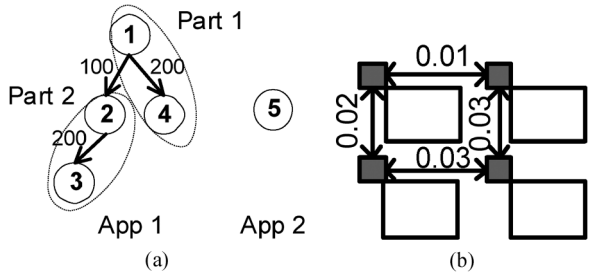


Fig. 12. Input task graph and network topology.

TABLE IV
TASK RUNTIME ON EACH CORE

	Task ID	Core a	Core b	Core c	Core d
Part 1,	1	5	1	5	5
App 1	4	5	5	1	5
Part 2,	2	1	5	1	1
App 1	3	5	1	5	1
App 2	5	1	5	5	1

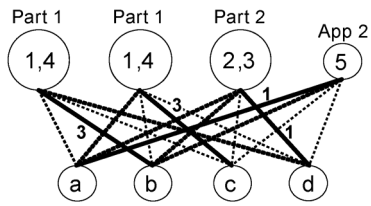


Fig. 13. Assigning cores to each sub graph group.

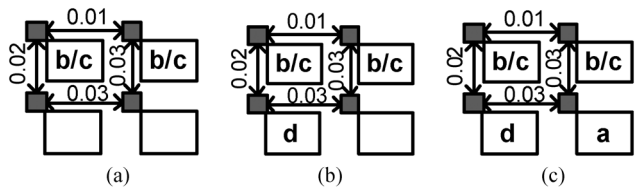


Fig. 14. Assigning tiles to each sub graph group.

Then, we get three sub task groups: *Part 1*, *Part 2* of *App 1* and *App 2*. The pseudo code in Fig. 11 is performed as follows.

a) The three groups are sorted in the order of *Part 1*, *Part 2* and *App 2* as in *Line 1* of Fig. 11.
b) Then, we budget the number of cores on each group. *Part 1* gets two cores while *Part 2* and *App 2* get one core.
c) A bipartite *Graph B* is then built up in Fig. 13 with the definition in *Line 3* of Fig. 11. Note that *Part 1* has a dummy node, since it gets two cores in the last step.
d) The minimal weighted bipartite matching algorithm is executed on *Graph B* to allocate cores to each group, like Fig. 13. Only the weights on picked edges are shown in Fig. 13.
e) From *Line 5* to *Line 12*, we will allocate network tiles to the three groups. First, *Part 1*, *Part 2* and *App 2* are sorted in order with total communication volume 200, 200, and 0. Then, *Part 1* with *Core b* and *c* is picked up. Two tiles with link number 0.01 are assigned to *Part 1*, as in Fig. 14(a). After that, *Part 2* and *App 2* are settled consequently in Fig. 14(b) and (c).

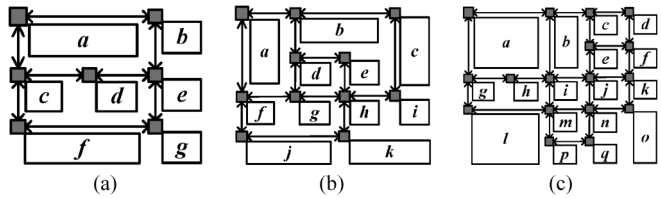


Fig. 15. Custom networks. (a) 7-core custom NOC. (b) 11-core custom NOC. (c) 17-core custom NOC.

After grouping the input task graph with cores and network tiles, we can run localized UNISM on *Part 1*, *Part 2* and *App 2* separately.

VII. EXPERIMENTAL RESULTS

A. Experiment Setup

In this section, we use *E3S* benchmark suite from the industry [25] and several random benchmarks by *TGFF-3.5* [26]. All the methods are implemented in C++. *hMetis v1.53* is used for min-cut partitioning [27]. *Gurobi v2.02* is adopted as the MILP solver [28].

As mentioned in [21], authors implemented the first and latest sequential flow A3MAP-GA (solved by Genetic Algorithm) supporting all the architectures of NOC (see Fig. 2). A min-cut partitioning is employed to translate the task graph into a core graph. In [21], they also expanded NMAP in [18] to support all the network architectures. Comparisons will be made between these two previous works and our unified approach UNISM.

1) Task Graph: One group of task graphs is defined in the industrial E3S benchmark suite.

The other group is randomly generated by an open-source tool named *TGFF-3.5*. However in *TGFF*, the communication volume, runtime and energy consumption should be provided by users. In our implementation, all these parameters are set similarly with *E3S* benchmarks. For a single task, the maximum difference on energy consumption could be 10 times in worst cases, if it is scheduled on different cores.

2) Network Topology: Besides task graphs, the other important input is network topology, the latency and energy to transfer one bit on each link.

For network topology, regular mesh, irregular mesh and custom NOC are adopted. For these custom networks in Fig. 155, all the tiles are randomly placed according to their areas and each tile consumes one router, which is the same as in [21].

For latency and energy, we are short of industrial data on this part. Fortunately, as pointed in [29], about 25% of total energy is spent on the networks. So, the practical ratio between core computing energy and communication energy should be around 3:1. Be aware of this, we set the number on each link to keep this ratio and make our network settings practical. On the contrary, if the communication consumes a very large portion of total latency and energy, it would not be worthwhile to use NOC in our multi-processor design.

3) Metrics for Comparison: Two metrics are considered: the execution time of the given task graph (termed as $\boldsymbol{T_{exex}}$) and the overall energy (termed as $\boldsymbol{E_{total}}$). T_{exec} can be used to measure

TABLE V
(A) COMPARISONS ON 3 × 3 MESH. (B) COMPARISONS ON 3 × 3 IRREGULAR MESH. (C) COMPARISONS ON A 7-CORE CUSTOM NOC

(a)

Benchmark	NMAP [18]			A3MAP-GA [21]			UNISM			Accelerated UNISM		
	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)
auto-indust	21.5	0.56	0.03	21.0	0.55	0.03	16.5	0.57	384	16.5	0.57	2
consumer	8833	346	0.04	8320	315	0.04	6540	325	391	6780	320	7
networking	11245	354	0.03	11245	354	0.03	11245	354	289	11245	354	5
telecom	14	0.50	0.03	13	0.50	0.03	12	0.52	457	12	0.52	4
Normalized	1.19	1.01	0.009	1.15	0.98	0.009	0.99	1	105.0	1	1	1

(b)

Benchmark	NMAP [18]			A3MAP-GA [21]			UNISM			Accelerated UNISM		
	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)
auto-indust	40.5	1.10	0.03	38.2	1.11	0.03	32.1	0.87	353	34.9	0.86	8
consumer	8361	390	0.03	7151	370	0.04	6341	313	412	6368	301	8
networking	14282	506	0.03	11785	543	0.03	11905	383	398	11924	406	9
telecom	19.1	0.67	0.03	17.8	0.66	0.03	15	0.52	434	16	0.55	7
Normalized	1.22	1.26	0.004	1.08	1.26	0.004	0.96	0.99	50.46	1	1	1

(c)

Benchmark	NMAP [18]			A3MAP-GA [21]			UNISM			Accelerated UNISM		
	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)
auto-indust	45.2	1.03	0.01	43.0	1.01	0.02	34.5	0.96	278	35.6	0.99	2
consumer	9500	312	0.02	8500	265	0.01	6950	223	324	7000	220	5
networking	10773	285	0.01	8288	256	0.02	7588	211	483	7881	226	6
telecom	37	0.85	0.01	36	0.91	0.02	27	0.85	439	30	0.80	5
Normalized	1.31	1.20	0.003	1.17	1.12	0.005	0.96	0.99	93.03	1	1	1

the performance. Less T_{exec} means the task graph could be finished with shorter time, which equals a higher performance of the system. For fair comparison, the idea of one dynamic routing algorithm in [30] is implemented supporting different NOC architectures, which is used to simulate the real-time behavior of the task graph and calculate T_{exec} and E_{total} when scheduling and mapping are finished.

B. Comparisons on Industrial Benchmarks

First, an industrial benchmark suite named *E3S* is tested. We adopt a 3 × 3 regular mesh with nine homogeneous cores, a 3 × 3 irregular mesh with nine heterogeneous cores and a seven-core custom NOC. The custom network is shown in Fig. 15(a).

Results on regular mesh are listed in Table V(a), which shows a 19% and 15% improvement on the execution time, compared with NMAP and A3MAP-GA. Meanwhile, the energy consumption is similar in these three works. We also find larger improvement on performance/energy (i.e., 26.5% and 12.5% less execution time with 23% and 19% lower energy) for irregular mesh and custom NOC on average. However, there is one exception. For *networking* benchmark in Table V(b), our method cannot make a shorter T_{exec} compared with A3MAP-GA. This is because a much better E_{total} is achieved only by a small increase on T_{exec}, which has a lower total cost in (1).

Meanwhile, we also list the results of UNISM without acceleration to show the global minimum found by the solver, if the runtime is acceptable. We can find that our acceleration heuristics can get almost the same result in Table V(a) and an acceptable tradeoff between the solution quality and runtime in Table V(b) and (c). For the case *telecom* in Table V(c), our accelerated method even gets lower energy. But the total cost in (1) is higher than the non-accelerated method, because T_{exec} is higher.

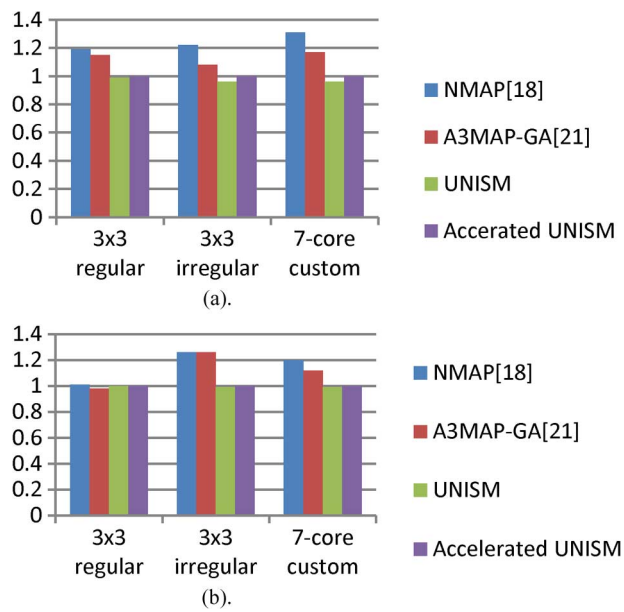


Fig. 16. Comparisons on industrial benchmarks. (a). Improvement on execution time T_{exec}; (b). improvement on total energy E_{total}.

The CPU time of UNISM is much larger than A3MAP-GA and NMAP, even if localized unification of scheduling and mapping is applied for runtime acceleration. However, the runtime of our algorithm is scalable for larger benchmarks, as in Section VII-D.

At last, improvements on execution time T_{exec} and total energy E_{total} are drawn in Fig. 16(a) and (b), respectively, including different types of NOC.

TABLE VI
(A) COMPARISONS ON 4 × 4 AND 5 × 5 REGULAR MESH. (B) COMPARISONS ON 4 × 4 AND 5 × 5 IRREGULAR MESH. (C) COMPARISONS ON 11-CORE AND 17-CORE CUSTOM NOC

(a)

Network	Benchmark	NMAP [18]			A3MAP-GA [21]			Accelerated UNISM		
		T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)
4x4 Mesh	tg30	374	24.6	0.04	362	24.4	0.05	333	24.8	21
	tg50	435	34.0	0.06	421	33.9	0.23	373	34.0	34
5x5 Mesh	tg50	525	40.5	0.08	514	40.1	1.03	484	41.2	54
	tg70	659	67.9	0.17	638	67.3	1.84	609	69.3	81
Normalized		1.11	0.99	0.002	1.08	0.98	0.13	1	1	1

(b)

Network	Benchmark	NMAP [18]			A3MAP-GA [21]			Accelerated UNISM		
		T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)
4x4 Irregular	tg30	432	32.1	0.04	356	31.5	0.05	293	24.1	25
	tg50	572	57.4	0.06	506	56.2	0.22	468	52.6	40
5x5 Irregular	tg50	452	49.2	0.08	435	43.3	1.38	382	35.2	49
	tg70	663	81.5	0.17	553	73.9	2.02	512	65.0	85
Normalized		1.29	1.27	0.002	1.13	1.18	0.15	1	1	1

(c)

Network	Benchmark	NMAP [18]			A3MAP-GA [21]			Accelerated UNISM		
		T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)	T_{exec}	E_{total}	CPU (s)
11-core Custom	tg30	473	34.2	0.03	455	31.1	0.11	375	26.5	20
	tg50	504	40.2	0.04	478	38.7	0.13	393	31.6	27
17-core Custom	tg50	497	48.9	0.04	453	47.5	0.21	369	38.6	41
	tg70	526	61.4	0.06	501	59.7	0.24	432	53.5	52
Normalized		1.27	1.24	0.001	1.20	1.18	0.005	1	1	1

C. Comparison on Random Benchmarks

Then, we generate three groups of larger benchmarks, which contains no fewer than 30, 50, and 70 tasks by *TGFF-3.5*. We tested five random benchmarks for each group. Correspondingly, larger networks of 4 × 4 and 5 × 5 tiles are adopted as well. The custom networks are shown in Fig. 15(b) and (c).

Results are listed in Table VI(a), (b), and (c). In Table VI, we do not use our method without acceleration, because the runtime of UNISM grows exponentially. Therefore, in this case, our acceleration technique becomes necessary when the original MILP model ends up with unacceptable CPU time.

On these larger benchmarks of Table VI, we can still find that accelerated UNISM works better than NMAP and A3MAP-GA (22% and 14% less $T_{\rm exec}$ with 17% and 11% lower $E_{\rm total}$ on the average of all the network architectures), even if some solution degradation is introduced by our accelerating heuristics.

Also, improvements on execution time and energy are shown Fig. 17(a) and (b) for different types of NOC.

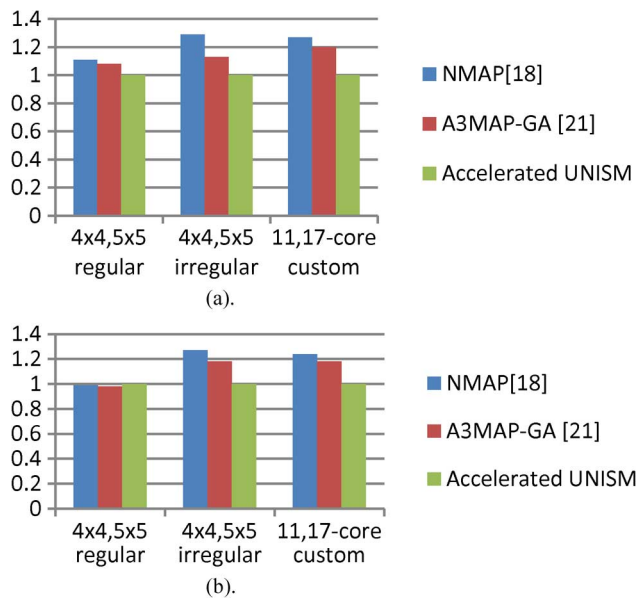


Fig. 17. Comparisons on random benchmarks. (a) Improvement on execution time $T_{\rm exec}$. (b) improvement on total energy $E_{\rm total}$.

D. Comparisons on Runtime

At last, we generate very large benchmarks with more than 150 tasks. Correspondingly, networks with more tiles are used to test the runtime of NMAP, A3MAP-GA and our method. The result is shown in Fig. 18. The runtime of our method (blue line) grows almost linearly as the network gets larger. The reason for this linear growth is that UNISM is only performed locally using the heuristics in Section VI, even if the MILP problem itself has non-polynomial complexity.

Meanwhile, the straightforward implementations of NMAP and A3MAP-GA have a near-exponential grow. However, our localization procedure in Section VI can also be applied on these two algorithms, which will give them a linear scalability as well. But the solution quality of NMAP and A3MAP-GA will be further diminished.

E. Sensitivity Analysis on Equation (1)

In this section, we study how the results will be affected if the weight factors in (1) are changed. The benchmark *tg50* is performed on 4 × 4 irregular mesh and the benchmark *tg70* is performed on 17-core custom networks.

In (1), α_1 is related to total energy while α_0 and α_2 are related to execution time. In order to study the tradeoff between energy

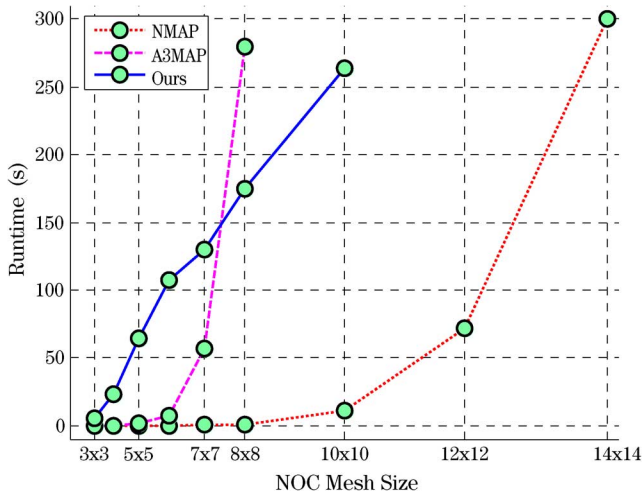


Fig. 18. Runtime on larger benchmarks.

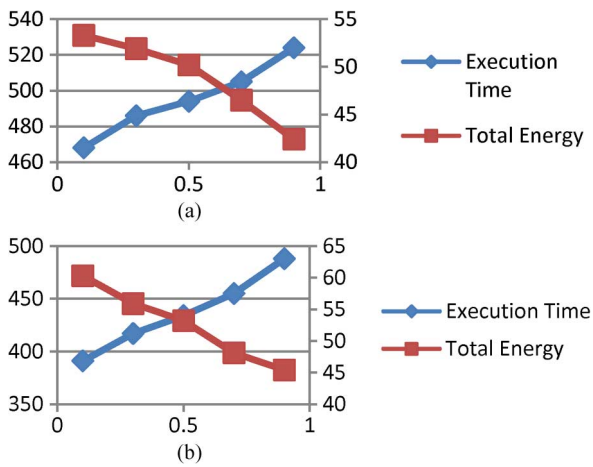


Fig. 19. Sensitivity analysis on (1). (a) *tg50* on 4 × 4 irregular mesh. (b) *tg70* on 17-core custom NOC.

and performance, α_1 is set to 0.1, 0.3, 0.5, 0.7, and 0.9. Meanwhile, we still set $\alpha_0 = 10\alpha_2$. So, α_0 would be 0.81, 0.63, 0.45, 0.27, and 0.09. The result is shown in Fig. 19, where we can see the different tradeoffs between T_{exec} and E_{total}. However, for different benchmarks, detailed shape of the curve could be different.

VIII. Conclusion and Future Work

A. Conclusions

In this paper, a unified flow combining task scheduling and core mapping named UNISM is proposed to support regular mesh, irregular mesh and custom NOC, using MILP. To enable this MILP modeling on irregular and custom NOC, a novel graph model called Labeled Graph is developed to calculate the communication latency and energy. Moreover, this model does not introduce any new variables, which makes our unified model as easy as a single scheduling algorithm in terms of the number of variables in MILP. Then, we accelerate our UNISM using NOC localization. Compared with two latest previous works, experimental results show that 15% and 11.5% improvement

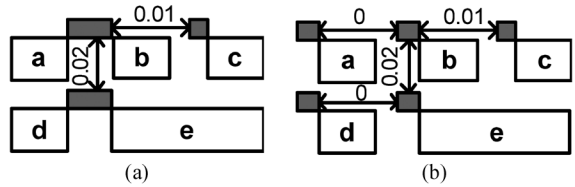


Fig. 20. Shared routers by multiple tiles.

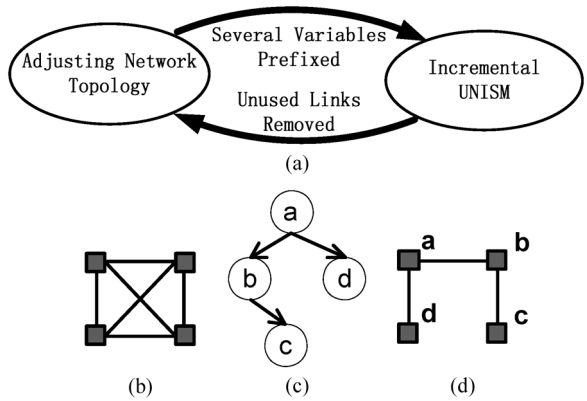


Fig. 21. Cooperation with topology generation.

on the execution time of the task graph is achieved on average with similar energy consumption for regular mesh NOC. For irregular mesh and custom NOC, the improvement is 27.3% and 14.5% with 24.3% and 18.5% lower energy on average. Moreover, our method is scalable in terms of runtime.

B. Future Extensions

1) Shared Routers by Multiple Tiles: In all the test cases of this paper, each tile owns one router. However, in custom NOC design, it is common to have shared routers by multiple tiles, as in Fig. 20(a). In this situation, we can generate an equivalent topology by adding some dummy routers like Fig. 20(b). After that, Labeled Graph and UNISM still work.

2) Cooperation With Topology Generation: By far, our algorithm takes network topology as an input. More benefit will be achieved if we do a co-synthesis combining network topology generation and our task/core/tile assignment. Compared with scheduling and mapping, topology generation is more discrete which is harder to build an analytical model and integrate with UNISM. Meanwhile, this integration will cause an explosion of the solution space even if we could take these two issues together.

Another practical choice is to co-act topology generation and UNISM incrementally. A general idea is shown in Fig. 21(a).

An example is illustrated to show how this idea works. First, a complicated clique topology is initiated as in Fig. 21(b). Then, a task graph is given in Fig. 21(c). After that, it is scheduled and mapped onto this topology and three links could be reduced without decreasing the system performance, as in Fig. 21(d). For larger systems, more iterations will be needed to make a balance between system cost and performance. This methodology will be suitable in custom design, e.g., application-specific NOC.

3) Cooperation With Routing: Moreover, better estimation of the routing congestion can be achieved in the scheduling and

mapping stage, if our algorithm is aware of the specified routing strategy.

REFERENCES

[1] D. Shin and J. Kim, "Power-aware communication optimization for networks-on-chip with voltage scalable links," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, 2004, pp. 170–175.

[2] H. C. Chi, C. M. Wu, and J. H. Lee, "Integrated mapping and scheduling for circuit-switched network-on-chip architectures," in *Proc. 4th IEEE Int. Symp. Electron. Design, Test, Appl. (DELTA)*, pp. 415–420.

[3] H. Yu, Y. Ha, and B. Veeravalli, "Communication-aware application mapping and scheduling for NoC-based MPSoCs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 3232–3235.

[4] P. Ghosh, A. Sen, and A. Hall, "Energy efficient application mapping to NoC processing elements operating at multiple voltage levels," in *Proc. ACM/IEEE Int. Symp. Netw. Chip*, 2009, pp. 80–85.

[5] R. Marculecu, U. Y. Ogras, L. S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: Circuit, microarchitecture, and system-level perspectives," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 1, pp. 3–21, Jan. 2009.

[6] G. Varatkar and R. Marculescu, "Communication-aware task scheduling and voltage selection for total systems energy minimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2003, pp. 510–517.

[7] V. Kianzad, S. S. Bhattacharyya, and G. Qu, "CASPER: An integrated energy-driven approach for task graph scheduling on distributed embedded systems," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Arch. Processors*, 2005, pp. 191–197.

[8] P. C. Chang, I. W. Wu, J. J. Shann, and C. P. Chung, "ETAHM: An energy-aware task allocation algorithm for heterogeneous multiprocessor," in *Proc. Design Autom. Conf.*, 2008, pp. 776–779.

[9] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proc. Conf. Design, Autom., Test Eur.*, 2004, pp. 16–20.

[10] C. L. Chou and R. Marculescu, "User-aware dynamic task allocation in networks-on-chip," presented at the Conf. Design, Autom., Test Eur., Munich, Germany, 2008.

[11] M. A. Al Faruque, R. Krist, and J. Henkel, "ADAM: Run-time agent-based distributed application mapping for on-chip communication," in *Proc. Design Autom. Conf.*, 2008, pp. 760–765.

[12] M. Kandemir, O. Ozturk, and S. P. Muralidhara, "Dynamic thread and data mapping for NoC based CMPs," in *Proc. Design Autom. Conf.*, 2008, pp. 852–857.

[13] G. Chen, F. Li, S. W. Son, and M. Kandemir, "Application mapping for chip multiprocessor," in *Proc. Design Autom. Conf.*, 2008, pp. 620–625.

[14] C. Ostler and K. S. Chatha, "An ILP formulation for system-level application mapping on network processor architectures," in *Proc. Conf. Design, Autom., Test Eur.*, 2007, pp. 99–104.

[15] M. Kandemir, O. Ozturk, and V. S. R. Degalahal, "Enhancing locality in two-dimensional space through integrated computation and data mappings," in *Proc. 20th Int. Conf. VLSI Design*, 2007, pp. 227–232.

[16] S. Tosun, O. Ozturk, and M. Ozen, "An ILP formulation for application mapping onto network-on-chips," in *Proc. Int. Conf. Appl. Inform. Commun. Technol. (AICT)*, 2009, pp. 1–5.

[17] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proc. Asia South Pacific Design Autom. Conf.*, 2003, pp. 233–239.

[18] S. Murali and G. De Micheli, "Bandwidth-constrained mapping of cores onto NoC architecture," in *Proc. Conf. Design, Autom., Test Eur.*, 2004, pp. 896–901.

[19] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2006, pp. 355–362.

[20] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 407–420, Apr. 2006.

[21] W. Jang and D. Z. Pan, "A3MAP: Architecture-aware analytic mapping for networks-on-chip," in *Proc. Asia South Pacific Design Autom. Conf.*, 2010 , pp. 523–528.

[22] K. Srinivasan, K. S., and Chatha , "A technique for low energy mapping and routing in network-on-chip architectures," presented at the Int. Symp. Low Power Electron. Design, San Diego, CA, 2005.

[23] C. Seiculescu, S. Murali, L. Benini, and G. De Micheli, "NoC topology synthesis for supporting shutdown of voltage islands in SoCs," in *Proc. Design Autom. Conf.*, 2009, pp. 822–825.

[24] G. Leary and K. S. Chatha, "Automated technique for design of NoC with minimal communication latency," in *Proc. 7th IEEE/ACM Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, 2009, pp. 471–480.

[25] R. Dick, "E3S Benchmark," [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/

[26] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. 6th Int. Workshop Hardw./Softw. Codesign*, 1998, pp. 97–101.

[27] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. Design Autom. Conf.*, 1999, pp. 343–348.

[28] Gurobi Optimization, Houston, TX, "Gurobi Solver," 2009. [Online]. Available: http://www.gurobi.com/html/academic.html

[29] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Sep. 2007.

[30] J. Hu and R. Marculescu, "Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures," in *Proc. Conf. Design, Autom., Test Eur.*, 2003, pp. 688–693.

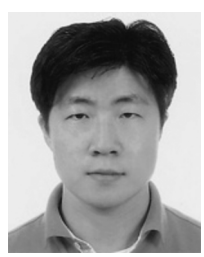
Ou He received the B.S. degree from the Department of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an, China, in 2006. He is currently pursuing the Ph.D. degree from EDA Labs, Department of Computer Science and Technology, Tsinghua University, Beijing, China.

From 2009 to 2010, he was a visiting scholar in the University of Texas, Austin. He is currently with IBM Microelectronics, China Design Center, STG, Beijing, China, where he works on SOC/IP design. His current research interest is floorplanning and high-level synthesis for NOC-based SOC.

Mr. He was the recipient of an annual scholarship for excellent students for Tsinghua University from 2006 to 2009 and Cadence Scholarship (First Class) in 2008. As a team member, he was a recipient of three awards in The Mathematical Contest in Modeling (MCM), China Undergraduate Mathematical Contest in Modeling (CUMCM), and China National Undergraduate Electronic Design Contest (NUEDC).

Sheqin Dong received the B.S. degree (with highest honors) in computer science, the M.S. degree in semiconductor physics and devices, and the Ph.D. degree in mechatronic control and automation from the Harbin Institute of Technology, Harbin, China, in 1985, 1988, and 1996, respectively.

From 1997 to 1999, he worked as a Post-Doctoral Fellow with The State Key Laboratory of Computer-Aided Design and Computer Graphics, Zhejiang University, Hangzhou, China. He is currently an Associate Professor and the Director of the EDA Laboratory, Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include computer-aided design for very large scale integration, parallel algorithms, multimedia application specific integrated circuits, and hardware design.



Wooyoung Jang (S'08) received the B.E. degree in radio science and technology from Kyung Hee University, Suwon, South Korea, in 1998, the M.S. degree in electrical and computer engineering from Yonsei University, Seoul, South Korea, in 2000 and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, in 2011.

Since 2000, he has been with System Large Scale Integration Division, Samsung Electronics, Suwon, South Korea, as a Senior Engineer. His current research interests include computer architecture and nanometer physical design for on-chip communication.

Mr. Jang was the recipient of the SK Telecom Scholarship for 1994–1997, the Samsung Outstanding Achievement Award in 2005, and the Samsung Scholarship for 2006–2011.

Jinian Bian (M'05) received the degree from Tsinghua University, Beijing, China, in 1970.

Since 1970, he joined Tsinghua University, where he is currently a Professor with the Department of Computer Science and Technology. His current research interests include SOC-oriented design methodology and system level synthesis and verification.

Dr. Bian has joined several National Key Projects in China, including the PANDA VLSI CAD System which received the First Class National Science and Technology Award.

David Z. Pan (S'97–M'00–SM'06) received the Ph.D. degree in computer science from the University of California, Los Angeles (UCLA), in 2000.

From 2000 to 2003, he was a Research Staff Member with the IBM T. J. Watson Research Center. He is currently an Associate Professor and the Director of the Design Automation Laboratory, Department of Electrical and Computer Engineering, University of Texas, Austin. His current research interests include nanometer VLSI physical design, design for manufacturing, vertical integration of technology, design and architecture, and design/CAD for emerging technologies. He has published over 140 refereed papers in international conferences and journals, and holds eight U.S. patents.

Dr. Pan has served as an Associate Editor for four premier IEEE journals, including IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS (since 2006), IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS (since 2007), IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS (2008–2009), and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS (2006–2007). He has served in the committees of many major VLSI/CAD Conferences, including ASPDAC (Subcommittee Chair), DAC (Subcommittee Chair), DATE, ICCAD (Subcommittee Chair), ISPD (Program/General Chair), among many others. He was a recipient of a number of awards, including the ACM/SIGDA Outstanding New Faculty Award (2005), NSF CAREER Award (2007), UCLA Engineering Distinguished Young Alumnus Award (2009), SRC Inventor Recognition Award three times (2000 and 2008), IBM Faculty Award four times (2004–2006, 2010), 6 Best Paper Awards (BPA), and many other BPA nominations at top conferences such as DAC, ICCAD, ASPDAC, DATE, ISPD, and SRC Techcon. He is an IEEE CAS Society Distinguished Lecturer for 2008–2009.