A Scalable and Adaptable ILP-Based Approach for Task Mapping on MPSoC Considering Load Balance and Communication Optimization

Kai Huang, Xiaomeng Zhang, Dandan Zheng, Min Yu, Xiaowen Jiang, Xiaolang Yan, Lisane B.de Brisolara, and Ahmed Amine Jerraya

***Abstract*—Task mapping has been a hot topic in MPSoC software design for decades. During the mapping process, load balance and communication optimization have been two important performance optimization factors. This paper studies the relations between load balance, inter-processor communications and communication pipeline technique during the mapping process, and proposes an Integer Linear Programming (ILP)-based static task mapping approach, which considers both load balance and communication optimization. The approach consists of an optimized ILP model for task mapping with fewer variables compared to previous ILP mapping works. Moreover, to enhance the scalability of the ILP task mapping, the Task-Processor-Cluster (TP-CLUSTER) algorithm is proposed to reduce the scale of the task graph and the number of processors and then solve the coarse-grained input by the ILP mapping. To increase the adaptability of the ILP task mapping, the improved augmented ϵ-constraint (AUGMECON2) method is further integrated with the ILP formulations to select the best mapping for different applications. Experimental results on a 2/4/8/16/24-CPU platform of both synthetic and real-life benchmarks demonstrate the efficiency of the proposed approach.**

***Index Terms*—Integer Linear Programming, multi-objective optimization, MPSoC, graph partitioning, task mapping.**

I. Introduction

MULTIPROCESSOR System-on-Chip (MPSoC) has become a necessary part of embedded systems due to the ever-growing functional and performance requirements. To achieve optimal performance when running software applications on MPSoC systems, hardware and software factors should be both considered. Among them, building an application into dependent or independent tasks, mapping these tasks onto multiple appropriate processors, and scheduling the tasks according to certain policies, i.e. task partitioning, mapping, and scheduling are key performance factors.

Task mapping has been studied for years due to its direct impact on system performance. With the increasing number of processors and complicated communication relations between processors, load balance and inter-processor communication optimization are becoming more important performance optimization factors. Lots of researches have been done on load balance [1], [2] and communication optimization [3]–[7] for MPSoC. However, during the mapping process, as minimizing communication overheads tends to assign tasks to only a small number of processors and load balancing tends to distribute tasks evenly among all the processors [1], these two optimization objectives may require a compromise to achieve better performance according to the communication and computation features of the input applications.

Of the two factors, communication optimization-related works can be divided into two kinds, i.e. application-level and architecture-level works. The former usually takes communications as abstract values decided by the amount of data transfer, while the latter takes communications as specific components, like memory-related issues [8], etc. This paper targets at application-level communication optimization. Existing mapping methods consider application-level communication optimization mostly by mapping heavily-communicating tasks as close as possible [3], [5], [9] or duplicating key tasks to overlap communication and computation [7], [10], but few works [11] consider exploiting fine-grained communication optimization techniques, such as communication pipeline [6].

Communication pipeline preprocesses part of the communications beforehand, so communications and computations from different pipeline iterations can be overlapped to hide the communication time. Moreover, communication pipeline can be utilized in both acyclic and cyclic topologies with additional constraints for cyclic topologies. Compared to the communication optimization in previous mapping methods [3], [5], communication pipeline can effectively reduce the communication transfer time without increasing the computation loads. Therefore, it is beneficial to utilize communication pipeline during mapping. However, mapping and communication pipeline allocation are interrelated. Mapping decides the processor loads and inter-processor communications, which limits the usage of communication pipeline, while efficiently using communication pipeline requires the consideration of application topologies in mapping. Therefore, an in-depth study on task mapping considering communication pipeline allocation should be conducted.

As task mapping is an NP-hard problem and adopting communication pipeline adds another dimension to the problem,

Manuscript received on October xx, 2017. This research was supported by the grant 2017ZX01030-102-002 of National Science and Technology Major Project. Corresponding author is Dandan Zheng.

K. Huang, D. D. Zheng, and X. L. Yan are with the Institute of VLSI Design, Zhejiang University, Hangzhou, China, 310027 (e-mail:huangk,zhengdd,yan@vlsi.zju.edu.cn).

X. M. Zhang, M. Yu, and X. W. Jiang are also with the Institute of VLSI Design, Zhejiang University, Hangzhou, China, 310027(e-mail:xiaomeng_zhang,yu_min,xiaowen_jiang@zju.edu.cn).

L. B. Brisolara is with Federal University of Pelotas, Brazil (email: lisane@inf.ufpel.edu.br).

A. A. Jerraya is with CEA LETI, MINATEC, Grenoble Cedex F38054, France (email: ahmed.jerraya@cea.fr).

Integer Linear Programming (ILP)-based methods are used due to its high quality solutions and easy modeling. However, in previous ILP-based mapping approaches [5], [12] aiming at high performance, the mapping problem is usually formulated by two key binary variables: the mapping relation of a task and each processor and the mapping relation of each pair of tasks and each processor. Therefore, the total number of variables is quadratic to the number of tasks, leading to a rapidly increasing problem scale as the number of tasks increases. Therefore, to exploit the ILP-based methods, it is necessary to build a better ILP mapping model with fewer variables and lay a foundation for further ILP optimizations.

Observing the above problems, we propose a static task mapping approach for performance improvements for MPSoC. The approach contains an optimized ILP model to solve the task mapping problem, considering load balance, inter-processor communication minimization, and communication pipeline allocation. It models computation and communication mapping separately, which can largely reduces the number of variables with no impact on ILP accuracy, especially suitable for sparse applications on bus-based MPSoC.

Moreover, when investigating the ILP-based methods to find optimal mapping considering load balance and communication optimization, we are faced with two further challenges.

1) Scalability. Although the optimized ILP model reduces variables, it is still inevitable that the solving time of ILP increases with the growing scale of the system.
2) Adaptability. As load balance and communication optimization may affect one another, it is unclear that how to determine their importance to obtain better results for applications with different computation and communication features.

Based on the optimized ILP mapping model, the two challenges are further investigated and the solutions are proposed. To deal with the scalability challenge, we first analyze the complexity of the ILP variables, and develop a Task-Processor-Cluster (TP-CLUSTER) algorithm, which combines fine tasks into coarse tasks and partitions processors into clusters, and uses these "coarse" inputs to find ILP results. To handle the adaptability challenge, we integrate the improved augmented ϵ-constraint method (AUGMECON2) [13] with our ILP formulations to generate a set of optimal solutions under different computation and communication features and choose the mapping with best performance for the input application.

This paper has the following three main contributions:

- We propose a new ILP model to solve the task mapping problem considering load balance and communication optimization targeting at performance improvements. To appropriately exploit communication pipeline into task mapping, we first analyze the relationship between communication pipeline, load balance, and inter-processor communication minimization, and then build an ILP model considering the three factors.
- To increase the scalability of the proposed ILP model, we develop a TP-CLUSTER algorithm. The algorithm first combines fine tasks into coarse ones and partitions processors into clusters to obtain a smaller-scale ILP input, then uses the proposed ILP model to find mapping and communication pipeline allocation.
- Although our target is to obtain minimum execution time, we have considered communication pipeline allocation, load balance, and inter-processor communication minimization sub-objectives. To increase the adaptability of the proposed ILP, we exploit a multi-objective-based way to formulate the problem and integrate the multi-objective optimization method AUGMECON2 with the ILP formulations to get optimal results for applications with different computation and communication features.

The rest of the paper is organized as follows. Section II gives some related works. Section III presents the theoretical foundation. Section IV introduces the proposed ILP-based mapping model. Section V and Section VI describes the scalable and adaptable extension based on the ILP model respectively. Section VII discusses the experiments. Section VIII gives the conclusion and points out future directions.

II. Related Work

Task mapping has attracted much attention in the multiprocessor era due to its direct impact on performance factors. To obtain better system performance, a good static mapping is essential for two reasons: (1) as the first step of the synthesis of software and hardware, its quality directly affects the subsequent scheduling step as well as the performance; (2) to reduce the system overhead, a dynamic mapping is often conducted based on a good initial mapping [14]. Therefore, this paper propose a static task mapping approach to obtain performance improvements.

Existing static mapping approaches use various methods to find solutions [15], mainly including search-based and heuristic-based methods. Search-based methods consist of deterministic methods like ILP [5], [12], and meta-heuristics such as Simulated Annealing (SA) [16], Genetic Algorithm (GA) [17], Ant Colony Optimization (ACO) [18]. Among them, ILP-based methods have been applied due to its optimality, formulation flexibility, and availability of several solving tools. Liu et al. [12] proposed an ILP model to assign tasks on heterogeneous multiple processors minimizing system cost under time constraint. Tang et al. [19] presented an ILP-based mapping method to maximize task parallelism for dataflow graphs. Huang et al. [5] proposed an ILP-based static mapping and scheduling policy for optimal performance. Although search-based approaches can give high quality solutions, they may face the scalability issue when the problem instances get large. Moreover, the ILP-based methods mentioned above all utilize the task-processor mapping relation and task-pair-processor mapping relation as key variables, while none of them consider the communication-processors mapping relation variable as an optimization. Reference [20] proposed a design space exploration approach optimizing topology, process binding, and communication routing using a symbolic representation and linear constraints. Although this work and ours utilize similar modeling method, they aim at different problems with different considerations.

Contrary to search-based methods, heuristic-based methods have been developed for low time complexity but the solution

quality cannot be guaranteed. For homogeneous architectures, as applications can be described as directed graphs, mapping can be modeled as the graph partitioning problem. Among the graph partitioning algorithms, multilevel graph partitioning (MLGP) is the most widely used one for its relatively high quality and short computation time, and has been implemented in many libraries such as Metis [21] and Kahip [22]. Wu et al. [23] exploited Metis in their scheduling policy to partition applications for the multicore CPU-GPU architecture. Ahmad et al. [24] developed a simple graph partitioning algorithm to divide the tasks into several partitions with minimum inter-partition data movement and then map the partitions onto multiple processing units with pipelined schedule to minimize latency. Huynh et al. [25] applied MLGP to partition a stream graph to multiple GPUs to achieve best performance under a given memory constraint considering balanced workloads and communication overhead. Tosun [26] proposed a clustering algorithm for network-on-chip (NoC), which divides the processors into several clusters and the applications into several task clusters based on the modified Kernighan-Lin algorithm to reduce the input ILP scale. However, the above graph partitioning algorithms or mapping algorithms are on acyclic graphs and may not provide high quality on cyclic topologies, which greatly affects the efficient usage of communication pipeline. Moreover, although the idea of Tosun's algorithm is similar to ours, the design is based on NoC with tiles and routers, and cannot be directly used for embedded bus-based MPSoC architectures.

Moreover, load balance and communication optimization are important performance optimization factors during mapping and researches have been conducted on the contradicted relation of the two factors. Such mapping works can be divided into two categories. The first category is based on graph theories, where computation and communication are represented by vertices and edges of graphs, and both factors are handled by merging or measuring vertices and edges. A typical example is to use MLGP algorithms for mapping, where in the coarsening phase, tasks are combined according to their communications and load balance is set as a constraint. Kaushik et al. [27] proposed a run-time mapping strategy with a pre-processing phase merging graph vertices under defined communication and computation conditions for NoC-based MPSoC. Kang et al. [1] proposed a new cost function for static task mapping describing the conflict between communication minimization and computation balance, and then transformed the task mapping problem into the minimum N-cut problem. These methods are easy to implement and run fast, but they are inherently heuristics and the optimizing quality cannot be guaranteed. The second category is based on mathematical programming, mainly formulated by multi-objective optimizations. Cotton et al. [28] adopted the multi-criteria optimization by formulating several query strategies to find Pareto optimal solutions for multiprocessor mapping problems. Saidi et al. [29] formulated the multi-objective communication and computation tradeoff problem into a single-objective communication minimization with a computation balance constraint. In our previous work [5], we exploited a weighted sum multi-objective ILP formulations to represent the tradeoffs between load balance and communication cost reduction for processor mapping. In another work [11], we proposed an ILP mapping method for software pipeline-based MPSoC, considering load balance and communication pipeline. However, in [5] and [11] the weights are predefined by experience, which lacks the adaptability to different applications.

To solve the multi-objective optimization problems, popular methods include meta-heuristic-based methods, gradient-based methods, weighted sum methods, and ϵ-constraint-based methods. Meta-heuristic-based methods, especially genetic algorithm (GA)-based methods such as NSGA-II [30] are widely used to solve all kinds of multi-objective optimization problems, but such methods cannot explicitly handle constraints and also face the problem of large-scale variables. Gradient-based methods [31] can utilize the design sensitivity of objective functions and variables to adaptively determine the weight coefficients of multiple objectives, but it is not suitable for the mapping problem as the variables are most discrete and their gradients cannot be calculated. Weighted sum methods and ϵ-constraint-based methods are suitable for linear programming problems and easy to implement. The widely used Adaptive Weighted Sum (AWS) method [32] can generate evenly distributed Pareto solutions by adjusting the weights and adding constraints to narrow the solution space according to the lengths of the segments in objective space. The ϵ-constraint method [33] solves one of the N objective functions and transforms the remaining $N-1$ objective functions into constraints. Based on it, AUGMECON [34] and AUGMECON2 [13] further improve the efficiency of the ϵ-constraint method. Although weighting methods and ϵ-constraint-based methods can generate Pareto optimal points for our problem iteratively, the latter shows more advantages when integrating with ILP formulations [34] over the former. Therefore, ϵ-constraint-based methods have been integrated with integer, linear or integer linear problems [35], but have not been utilized to tackle the task mapping problem considering load balance and communication optimization.

This work proposes an ILP-based static mapping approach for best performance, considering both load balance and communication optimization. It also contains TP-CLUSTER to deal with the scalability issue, and integrates AUGMECON2 to tackle the adaptability issue of the multi-objective ILP.

III. Theoretical Foundation

A. Mapping model

The hardware multiprocessor system, denoted by $S(P)$, consists of n homogeneous processors (each denoted by $p \in P$) linked by point-to-point bus-based interconnections. Inter-processor communication is done through Direct Memory Access (DMA), so communication and computation can be overlapped. Intra-processor communication can be neglected due to its implementation by local memory.

A software application can be represented by a directed graph, denoted by $G(T, E, w_T, w_E)$. Each vertex represents one computation task of the application ($t \in T$) with weight w_T representing the computation time, and each edge represents one precedent dependency between a pair of tasks

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2859400, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

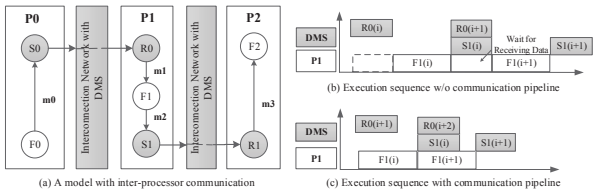


Fig. 1. An example of communication pipeline.

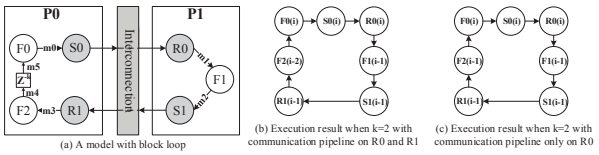


Fig. 2. Communication pipeline with cyclic topology.

($e \in E$) with weight w_E representing the communication transfer time. In this model, the directed graph may be acyclic or cyclic, which means that there may be a path starting from some vertex and ending with the same vertex. Such path is called as a task loop, and there will be a delay block inserted into the task loop to avoid deadlock. Moreover, an application will execute for many times, and we call the time that all tasks execute for once as one iteration.

Based on the system model and the application model, task mapping can be modeled as a mapping relation M between a graph set G and a processor set S denoted by $M : G \rightarrow S$, where each task from the targeted application is mapped onto its appropriate processor determining the computation workload on each processor, and precedent dependencies between tasks on different processors are mapped as inter-processor communications. To better illustrate the communication optimization technique, an inter-processor communication is further represented by a sending task and a receiving task with sending time w_S and receiving time $w_R (w_S + w_R = w_E)$.

B. Communication Pipeline

Communication pipeline is a fine-grained communication optimization technique, which can hide inter-thread communication transfer overhead. It takes the advantage of hardware like DMA, which can autonomously transfer data without the intervention of processors after initiation, to achieve the parallelization of computation and communication. Communication pipeline executes the data transfer for the computation of current iteration in advance and buffers the data in extra memory, so the subsequent computation can use the buffered data directly in the current iteration. As the example shows in Fig. 1(a), there is a chain of processors $P0$-$P2$, and on each processor let F be a computation task waiting for some input from receiving task R and outputting function results by sending task S to other processors. In Fig. 1(b), without communication pipeline, $F1$ has to wait for $R0$ of the same iteration. Nevertheless, in Fig. 1(c), since $R0$ is executed one iteration beforehand, $R0$, $F1$, and $S1$ of different iterations are executed in parallel with the help of communication pipeline, and the waiting time for receiving data can be saved. From the above analysis, we can observe that communication pipeline can totally hide inter-processor communication overhead when the communication time is no larger than its corresponding computation time. Therefore, this work focus on graphs with its communication-to-computation ratio (CCR) smaller than 1, which are commonly for real-life applications. Cases when $CCR > 1$ will be discussed in the future.

Based on the mechanism of communication pipeline, it can be applied to both acyclic and cyclic graphs. However, for acyclic graphs, communication pipeline can be used as much as possible, while for cyclic graphs, the usage of communication pipeline is limited by the number of delayed iterations in the task loop to avoid deadlock [6]. As the example in Fig. 2 demonstrates for $k = 2$, if $R0$ and $R1$ are both applied by communication pipeline, $F0(i)$ needs to receive data from $F2(i-2)$, and $k = 2$ leads to the data requirement of $F2(i-2)$ from $F0(i)$, which causes deadlock. Therefore, in a task loop with K delay iterations and M receiving blocks, we can preprocess part of the tasks before the iteration of the corresponding processor and apply communication pipeline to $min(K-1, M)$ receiving tasks. Readers can refer to [6] for more details about communication pipeline.

IV. The Proposed ILP-based Mapping

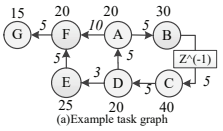
As discussed in Section III-B, communication pipeline can overlap the communication and computation to reduce inter-processor communication cost. Reference [6] has proved that adopting it for a determined mapping and scheduling can effectively improve performance. However, allocating communication pipeline during mapping poses several challenges. To better illustrate the challenges, we use processor load and inter-processor communication to represent mapping as they are direct reflections of the mapping result. Therefore, the challenges can be interpreted as the relations between processor load, inter-processor communication, and communication pipeline. This section first uses an example to illustrate the challenges, and then introduces the proposed ILP mapping model.

A. Motivational example

We aim to map a directed cyclic graph onto a 2-CPU platform as shown in Fig. 3 and its task execution time and communication time before mapping is shown beside the node and the edge (italic) of the graph. Right after the graph are 5 different mappings M0-M4 and their performance values, in which two or all of the three factors, load balance (LB), inter-processor communication minimization (ICM), and communication pipeline allocation (CP), are considered. From the performance results, we have the following observations:

1) Mapping considering communication pipeline can improve performance. M0 and M1 show the same task mapping but M1 considers communication pipeline, which gives smaller execution time than M0.
2) Load balance and communication pipeline allocation are interrelated. M1 shows a mapping with perfect load balance, but it exposes the cyclic relations $A \rightarrow B$ and $C \rightarrow D$ as inter-processor communications, which

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2859400, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems



(a)Example task graph

Set	LB	ICM	CP	Mapping	Performance
M0	√	√	×	P0: A,D,E,F; P1: B,C,G	185
M1	√	×	√	P0: A,D,E,F; P1: B,C,G; CP: F->G	183
M2	×	√	√	P0: A,B,C,D,E,F; P1: G; CP: F->G	176
M3	×	√	√	P0: A,B,C,D,E; P1: F,G; CP: A->F, E->F	153
M4	√	√	√	P0: A,B,C,D; P1: E,F,G; CP: A->F, D->E	144

(b)Different mappings

Fig. 3. A motivational example.

limits the application of communication pipeline. M4 trades off some load balance for more chances to allocate communication pipeline. Therefore, while load balance and communication pipeline allocation can be obtained simultaneously for acyclic graphs, the two factors require a tradeoff for cyclic graphs.

3) Inter-processor communication minimization and communication pipeline are strongly inter-dependent. M2 minimizes inter-processor communications but the effect of communication pipeline is not fully reflected because communication pipeline expects to expose more inter-processor communications. In M3, although inter-processor communications are not minimized, communication pipeline can be exploited more to improve performance. Therefore, inter-processor communication minimization and communication pipeline should be considered simultaneously.
4) Load balance and inter-processor communication minimization requires tradeoff. Compared to M4, M2 and M3 both has poor balance condition, which fail to exploit the parallelism of multiprocessors.

Therefore, to obtain high performance, we try to apply communication pipeline during task mapping, and our mapping should consider load balance, inter-processor communication minimization, and communication pipeline allocation.

B. ILP mapping model

Task mapping is an NP-hard problem, and exploiting communication pipeline adds another dimension to the problem. To obtain high-quality mapping considering load balance, inter-processor communication minimization, and communication pipeline allocation, we exploit ILP to solve the problem. As the three factors greatly affect the final performance, we build them into the objective function and set constraints for mapping relations and communication pipeline allocation. Like [28] and [19], we utilize pipelining style to execute different instantiations of tasks, so the objectives can approximate performance. As communication pipeline allocation requires the consideration of cyclic topologies, we first exploit Tarjan's algorithm [36] to find all task loops of the application graph, and then solve the ILP formulations.

Constants and variables: to better illustrate the ILP formulations, constants and variables of the model are first listed in Table I. In our model, there are two key binary variables $MT_{i,k}$ and $MC_{i,jk}$ representing the task-processor mapping relation and communication-processors mapping relation, respectively. While in the old model, task-processor mapping relation and task-pair-processor mapping relation are the key variables. Readers can refer to [5] for the old model. Compared to the old model, the new one has the following advantages:

TABLE I
CONSTANTS AND VARIABLES

Constant	Representation
$T/C/P/TL$	Task/communication/processor/task loop set
wc_i	Communication time of communication $c_i \in C$
wt_i	Average execution time of task $t_i \in T$
dly_m	The number of delayed iterations in task loop $tl_m \in TL$
$prt_{m,i}$	Communication-task loop relation: boolean constant, the value is 1 only if communication $c_i \in C$ is part of the task loop $tl_m \in TL$
$relation_{i,j}$	Communication-task relation: integer constant, the value is the starting task number when $j = 1$ and the ending task number when $j = 2$ of the communication $c_i \in C$
Variable	**Representation**
$MT_{i,k}$	Task-processor mapping relation: boolean variable, the value is 1 only if task $t_i \in T$ is mapped to processor $p_k \in P$
$MC_{i,jk}$	Communication-processor mapping relation: boolean variable, the value is 1 only if communication $c_i \in C$ is mapped between processors $p_j, p_k \in P$ $(j \neq k)$ or on the same processor $p_j/p_k (j = k)$
cp_i	Communication pipeline usage: boolean variable, the value is 1 only if communication pipeline is applied to communication $c_i \in C$

1) Variable reduction. With $MC_{i,jk}$, it is not necessary to use the task-pair-processor mapping relation to represent communication-processors mapping relation, and a constant $relation_{i,j}$ can represent the relation of $MT_{i,k}$ and $MC_{i,jk}$ instead. The scale of the variables taking the largest proportion of variable space is transformed from $|P||T|^2$ to $|C||P|^2$. For the same number of processors, the number of the variables of the new model increases slower than the old model. For sparse graphs with $|C| \ll |T|^2$, the new model has fewer variables than the old model when $|P|$ is relatively small. As sparse graphs are commonly seen in real-life applications, and in our proposed mapping $|P|$ will not be too large, the new model is effective for our problem.
2) Constant reduction. In the old model, communication is represented by a $|T|^2$ matrix and the matrix is mostly sparse with many 0s. In the new one, communication is represented by a $2|C|$ matrix consumed less memory.

Objective function: to achieve best performance, we convert load balance, inter-processor communication minimization, and communication pipeline allocation into minimizing the total computation workload gap of all processors(WG), the number of inter-processor communications(CN), and the left amount of inter-processor communication time after communication pipeline is applied(CP), given by (1). As possibly contradictory optimization factors, we assign weights λ_0, λ_1 and $(1-\lambda_0-\lambda_1)$ to the three objectives respectively. User can determine the weights by empirical values or other criteria. Note that for acyclic graphs, the weight of CP has no impact on ILP results, so communication pipeline can be allocated after mapping and CP weight is 0. For cyclic graphs, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2859400, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

formulations should all be utilized.

$$min(\lambda_0 \cdot WG + \lambda_1 \cdot CN + (1 - \lambda_0 - \lambda_1) \cdot CP) \quad (1)$$

The degree of load balance can be reflected from the similarity of the computation workload of all processors. WG is represented by the sum of the gap between each processor workload and the average computation workload of all processors.

$$WG = \Sigma_{k \leq |P|} || \Sigma_{i \leq |T|} (MT_{i,k} \cdot wt_i) - (\Sigma_{i \leq |T|} wt_i)/|P| || \quad (2)$$

For any communication, if it is mapped onto different processors, then it is an inter-processor communication. CN calculates the number of inter-processor communications.

$$CN = \Sigma_{i \leq |C|, j,k \leq |P|} (MC_{i,jk}) - \Sigma_{i \leq |C|, j \leq |P|} (MC_{i,jj}) \quad (3)$$

If communication pipeline is applied on a communication, then its transfer time is 0. Therefore, CP calculates inter-processor communication amount by removing communications mapped on the same processors and the communications saved by communication pipeline from the total communication amount.

$$\begin{aligned} CP = \Sigma_{i \leq |C|, j,k \leq |P|} (MC_{i,jk} \cdot wc_i) - \\ \Sigma_{i \leq |C|, j \leq |P|} (MC_{i,jj} \cdot wc_i) - \Sigma_{i \leq |C|} (cp_i \cdot wc_i) \end{aligned} \quad (4)$$

Constraints:

- Task-processor constraint: each task should be mapped to only one processor.

$$\Sigma_{k \leq |P|} MT_{i,k} = 1, \forall i \leq |T| \quad (5)$$

- Communication-processor constraint: each communication $c_i \in C$ should be mapped to only one pair of different processors or the same processor.

$$\Sigma_{j,k \leq |P|} MC_{i,jk} = 1, \forall i \leq |C| \quad (6)$$

- Task-communication relation constraint: for a communication $c_i \in C$ with starting task $t_m \in T$ and ending task $t_n \in T$, if t_m is mapped on processor $p_j \in P$, i.e. $MT_{m,j} = 1$ and t_n is mapped on processor $p_k \in P$, i.e.$MT_{n,k} = 1$, then $MC_{i,jk} = 1$.

$$MC_{i,jk} \leq (MT_{m,j} + MT_{n,k})/2 \quad (7)$$

$$MC_{i,jk} \geq MT_{m,j} + MT_{n,k} - 1, \forall i \leq |C|, j,k \leq |P| \quad (8)$$

- Communication pipeline location constraint: communication pipeline is only employed on inter-processor communications, i.e. communication pipeline should not be applied to any communication mapped on the same processor.

$$1 - cp_i \geq \Sigma_{j \leq |P|} MC_{i,jj}, \forall i \leq |C| \quad (9)$$

- Communication pipeline quantity constraint (only for cyclic graphs): in each task loop, the maximum number of communication pipeline used should be 1 less than the number of delayed iterations in this loop.

$$\Sigma_{i \leq |C|} (cp_i \cdot prt_{m,i}) \leq dly_m - 1, \forall m \leq |TL| \quad (10)$$

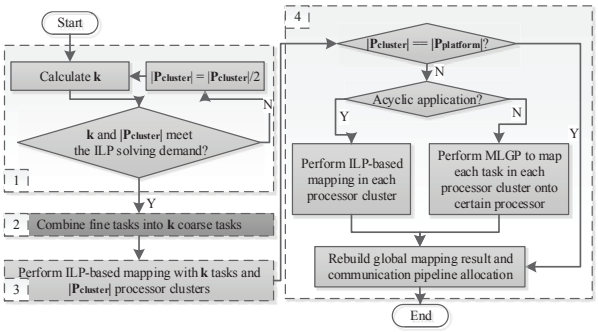


Fig. 4. The flow of TP-CLUSTER.

V. Scalable ILP-Based Task Mapping

A. Complexity Analysis

The scalability issue comes with the increasing scale of the system. Based on the above ILP formulations, we can notice that the number of the variables N can be calculated as

$$N = |T||P| + |C||P|^2 + |C| \quad (11)$$

Therefore, the problem scale increases with the increasing number of tasks $|T|$, communications $|C|$, and processors $|P|$. Among them, $|T|$ and $|C|$ may be of different orders of magnitudes for different applications, which become the bottleneck of ILP and are our main targets to reduce. For MPSoC, in most cases $|P|$ is less than 32 and reducing $|T|$ and $|C|$ can provide enough feasibility for ILP. However, when $|P|$ is relatively large, i.e. 16 for our experimental environment, reducing $|T|$ and $|C|$ is not enough to solve the scalability problem, which requires the reduction of $|P|$ as well. Therefore, we intend to tackle the scalability problem by first reducing $|T|$ and $|C|$ from the view of applications, and then if reducing $|T|$ and $|C|$ is not enough, we try to decrease $|P|$ from the view of platforms. Next subsection introduces the proposed TP-CLUSTER algorithm to solve the problem.

In addition, before utilizing the algorithm, users should determine N as the upper limit by experience. Since the solving time of ILP differs in different experimental environments and users have different endurance for computation time and memory, we cannot set a determined limit for N.

B. TP-CLUSTER Algorithm

1) Algorithm flow: The idea of TP-CLUSTER algorithm is to combine fine tasks into coarse tasks to simultaneously reduce $|T|$ and $|C|$, and if reducing $|T|$ and $|C|$ still cannot meet the demand of scalability, the algorithm bi-partitions the processors iteratively until the scalability demand is met. Fig. 4 shows the algorithm and we divide the flow into 4 phases.

- Phase 1 finds the proper number of coarse tasks k and processor clusters $|P_{cluster}|$ that can meet the ILP solving demand. We first assume $|P_{cluster}| = |P|$ and calculate k. As we hope to find the maximum k that meets the N limit under the specified $|P|$, we derive (12) to obtain k from (11). If the calculated k is larger than $|P|$, then we can allocate coarse tasks on all processors and each processor can have at least one task. In this way,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2859400, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

we can use the ILP-based mapping to obtain results. However, if the calculated k is smaller or equal to $|P|$, the constraint is too tight for mapping and we cannot obtain the communication pipeline allocation either. In this case, we decrease $|P|$ to half until k is larger than $|P_{cluster}|$. This phases continues until the ILP demand is met.

$$k = N/(\lceil(|C|/|T|)\rceil \cdot (|P|^2 + |P| + 1)), |C| > 0. \quad (12)$$

- Phase 2 combines the original fine tasks into k coarse tasks with the k and $|P_{cluster}|$ of Phase 1. We develop an algorithm to combine tasks considering load balance and communication optimization, especially focusing on the problems brought by communication pipeline. Details about this phase will be introduced in Section V-B2.
- Phase 3 performs the ILP-based mapping with the k coarse tasks and $|P_{cluster}|$ processor clusters using the ILP model in Section IV. Solving the ILP formulations, we can obtain the mapping of coarse tasks and the communication pipeline allocation between coarse tasks.
- Phase 4 further determines the mapping and communication pipeline allocation in each processor cluster and rebuilds results for the original task graphs. This phase first checks if the processors have been clustered. If not, then only tasks are combined in Phase 2, so the flow directly rebuilds the global mapping and communication pipeline allocation. Otherwise, this phase handles the coarse task mapping in each processor cluster according to the acyclic or cyclic topology of the whole task graphs. For acyclic graphs, allocating communication pipeline between processors in one cluster is independent of other clusters, which means the ILP-based mapping can be used in the cluster locally. However, for cyclic graphs, allocating communication pipeline between processors in one cluster should take task loops into account, which indicates the ILP-based mapping cannot be used in this case. Therefore, we utilize the original MLGP algorithm [21] in each cluster to obtain the mapping result, and communication pipeline is simply not used among such processors.

2) Phase 2 combining tasks: From the view of graph theory, the combination of fine tasks into coarse tasks fits the k-way partitioning problem, i.e. how to divide $|T|$ tasks into k partitions ($|P| \leq k \ll |T|$), and MLGP [21] is a fast and effective solution. Therefore, our method of combining tasks is proposed based on the general flow of MLGP and the detailed implementation is customized to the specific problem considering communication pipeline. To better illustrate our algorithm, we first introduce the general flow of MLGP and then describe our modified core algorithms.

a) General flow of MLGP: General MLGP includes coarsening and uncoarsening phases. The coarsening phase groups fine tasks into coarse ones through several levels until the number of the coarse tasks reaches the destination number k. A task which is not combined with others is called an unmatched task and once it finds another one to combine, the two tasks are matched. In each level, unmatched tasks are matched with adjacent ones according to *matching algorithms* to form the coarser graph by combining their computation time and communication relations, and the left unmatched tasks are copied to the coarser graph. The uncoarsening phase projects the k coarse tasks to the original graph by going through levels recorded in the coarsening phase. In each level, some fine tasks of this level is moved from its partition to one of its adjacent partitions according to *refinement algorithms* to refine the partition quality measured by communication and computation conditions. Moved tasks of this level will not be moved again, and the level finishes after no tasks to move.

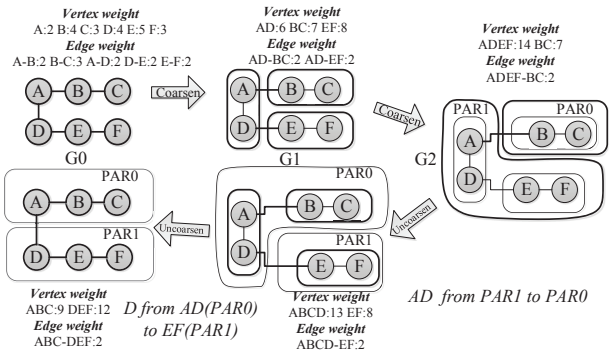


Fig. 5. An example of MLGP (6 tasks are partitioned into PAR0 and PAR1).

Fig. 5 gives an example of MLGP, which partitions G0 with 6 vertices into 2 parts. In the first level, A is matched with its adjacent D according to some matching algorithm and forms a coarser task AD, whose vertex weight is the sum of A and D's vertex weight and the edge relation is the combination of A and D's edge relations. The same goes for BC and EF, and G1 level completes. Since G1 has 3 tasks and has not reached the target number 2, G1 is further coarsened to G2 using AD, BC, and EF as finer tasks. As G2 has the target number of partitions, the coarsening phase ends and the uncoarsening phase begins from G1. AD of G1 originally belongs to PAR1, and it is moved to PAR0 to refine the partition quality. After there is no further refinement on G1, the uncoarsening goes on to G0. For refinement, D from PAR0 is moved to PAR1. Then no tasks in G0 can be moved for refinement and the whole MLGP ends. We can see the final results that PAR0 contains A,B, and C, and PAR1 contains D, E, and F.

b) Matching and refinement: From the MLGP flow, we see that matching algorithms greatly affect the coarsening phase and refinement algorithms largely decide the uncoarsening phase. Therefore, we propose our customized matching and refinement algorithms. As the conventional MLGP is developed on undirected graphs, applying the matching and refinement algorithms to directed graphs may introduce loops during coarsening and uncoarsening, which limits the usage of communication pipeline. Therefore, our matching and refinement algorithms consider directed edges and keep the original acyclic and cyclic dependency for communication pipeline allocation.

Our *matching algorithm* is based on Modified Heavy Edge Matching (HEM*) [21]. As shown in Fig. 6, our algorithm randomly chooses an unmatched vertex v, and matches it with one of its adjacent unmatched vertices u with maximum edge weights (line 1). For available tasks, the topology constraint

Require: A randomly chosen unmatched vertex v
Ensure: The adjacent task to be combined with
1: Find adjacent unmatched vertices of v with maximum edge weights, and add to set V_{maxe}
2: **for all** $u \in V_{maxe}$ **do**
3: **if** no delay between v and $u \wedge$ combining v and u not increase block loops **then**
4: Calculate W_{v_u}
5: **else**
6: $W_{v\_u} = -1$
7: Find u_{ready} from V_{maxe} with maximum W_{v_u}
8: **if** $W_{v\_u_{ready}}$ == -1 **then**
9: **return** NULL
10: **else**
11: **return** u_{ready}

Fig. 6. Matching algorithm.

is then checked (line 3) to make communication pipeline allocation on coarse tasks be interpreted as allocation on fine tasks. It first checks if delays exist between the two tasks to be combined and "no delays" ensures that the communications with delays are kept explicit and can be used in the coarse-task ILP formulations. It then checks if combining the two tasks will increase task loops and "not increase task loops" guarantees the left communications without delays between any pair of coarse tasks are of the same direction and communication pipeline can be applied directly. Finally, if there are still more than one available vertices, u with the maximum W_{v-u} is selected (line 7). W_{v-u} represents the sum of the weights of u's edges that connect u to vertices adjacent to v [21].

Our *refinement algorithm* is based on the greedy algorithm [21] but also considers the similar topology constraint as Fig. 6. As shown in Fig. 7, the algorithm aims to find a partition for a fine boundary task (connects to tasks in other partitions) to move to in order to improve the partition quality. The algorithm first uses balance constraint and topology constraint to select feasible partitions (line 3-4). Then, the fine boundary task is moved to the available partition which improves the communication condition most (line 6-8) or improves the balance condition when the communication condition cannot be improved (line 9-10).

To better quantify computations and communications, let w_v be the vertex weight of fine task v, W_i be the total vertex weights of partition i, and W_{min} and W_{max} be the minimum and maximum total vertex weights of any partition. Let $EO_i[v]$ be the number of edges starting from task v to partition i where v does not belong to, $EI_i[v]$ be the number of edges ending with task v from partition i where v does not belong to, $IO[v]$ be the number of edges starting from task v to other vertices in the same partition, and $II[v]$ be the number of edges ending with task v from other vertices in the same partition. Let ITR_{ij} be the number of edges starting from partition i and ending with partition j.

- The balance constraint limits that moving v from partition q to any partition p will not deteriorate the balance condition of each partition, which can be represented by $W_q - w_v > W_{min}, W_p + w_v < W_{max}$.
- The topology constraint limits that a fine boundary task can only move to the partition if there is no delays between itself and the fine tasks in that partition, and moving this task to the partition will not increase the number of inter-partition loops. An inter-partition loop ($v \rightarrow u$ and $x \rightarrow w$) is represented by two edges with opposite directions connecting different partitions, as shown in Fig. 8(a). An example in Fig. 8(b) shows the calculation process. Before moving, the number of inter-partition loops between partition q and partition p is $N_{before} = ITR_{qp} \cdot ITR_{pq}$. After moving, inter-partition edges connected to v are moved to the other partition and edges connected to v in the same partition become the inter-partition edges, and thus the number becomes $N_{after} = (ITR_{qp} - EO_p[v] + II[v]) \cdot (ITR_{pq} - EI_p[v] + IO[v])$. The constraint is $N_{before} \geq N_{after}$.
- To improve the communication condition, the number of inter-partition edges of vertex v should be reduced. The change of inter-partition edges can be represented by $\Delta ipe = (EI_p[v] + EO_p[v]) - (II[v] + IO[v])$.
- When communication condition cannot be improved, moving v should not worsen the balance of partitions, so the condition is $\Delta ipe == 0 \wedge W_q + W_p > w_v$.

Require: A fine boundary task v in partition q and all its adjacent partitions P
Ensure: v should be moved to which partition $p \in P$
1: $max\_ipe = 0; pmove = NULL$
2: **for all** $p \in P$ **do**
3: **if** Moving v to p violates the balance constraint $\vee$ violates the topology constraint **then**
4: continue
5: **else**
6: **if** Moving v to p reduces the number of inter-partition edges **then**
7: **if** $\Delta ipe \geq max\_ipe$ **then**
8: $max\_ipe = \Delta ipe$; $pmove = p$
9: **else if** Moving v to p keeps the number of inter-partition edges unchanged $\wedge$ moving v to p improves the balance condition **then**
10: $pmove = p$; break
11: **if** $pmove \neq NULL$ **then**
12: Move v to $pmove$

Fig. 7. Refinement algorithm.

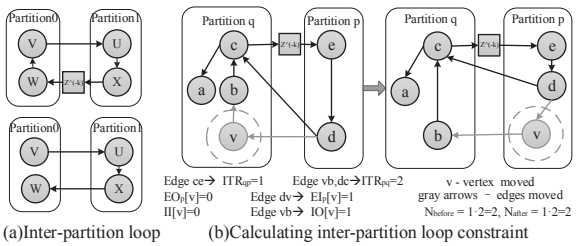


Fig. 8. Inter-partition loop.

VI. Adaptable ILP-based Task Mapping

The adaptability issue is originated from modeling the mapping problem using a multi-objective way. In the ILP model proposed in Section IV, setting λ_0 and λ_1 as fixed values narrows the solution space and affects the Pareto optimality of the multi-objective optimization. Through analysis and experiments, we have found that for acyclic graphs, only the weights of WG and CN impact the mapping results, while for cyclic graphs, WG, CN and CP all require tradeoff. To improve the optimality of the ILP-based mapping, we integrate a multi-objective optimization method AUGMECON2 with ILP to find Pareto optimal solutions. We first introduce AUGMECON2 in Section VI-A and integrate it with our ILP in Section VI-B.

A. AUGMECON2

Multi-objective optimizations deal with multiple possibly conflicting objectives to find a series of Pareto optimal solutions that cannot be improved in one objective function without deteriorating at least one of the rest [34]. The augmented ϵ-constraint method (AUGMECON) is a well-known method to solve the multi-objective problems, especially for multi-objective mathematical programming problems (MOMP). For problems with p objectives, AUGMECON chooses one objective function as the main objective and turns the remaining $p-1$ objectives into constraints. By varying the constraints, a set of Pareto optimal solutions can be obtained by solving the following mathematical programming problem [34]:

$$\begin{aligned} &max(f_1(x) + eps(a_2\frac{s_2}{r_2} + a_3\frac{s_3}{r_3} + ... + a_p\frac{s_p}{r_p})) \\ &st. f_2(x) - s_2 \geq e_2, ..., f_p(x) - s_p \geq e_p \end{aligned} \tag{13}$$

x is the vector of decision variables, $f_1(x), ..., f_p(x)$ are the p objective functions, e_i is the right hand side (RHS) for the specific iteration drawn from the grid points of the objective function $f_i(x)$, r_i is the range of the respective objective function $f_i(x)$, s_i is the surplus variable of the respective constraint ($s_i \in R^+$), and $eps \in [10^{-6}, 10^{-3}]$.

This work exploits the newly proposed AUGMECON2 to solve the multi-objective problem. Compared to AUGMECON, AUGMECON2 sets the coefficient $a_i = 10^{-(n-2)}$ ($n = 2, ..., p$) to produce a lexicographic order to optimize objective functions in case there are alternative optima, and checks the surplus value that corresponds to the innermost objective function in each iteration and thus a bypass coefficient $b = int(s_2/step_2)$ is added to reduce redundant iterations with the same optimal solutions. Here, $int()$ returns the integer part of a real number. $step_2 = r_2/q_2$, where q_2 represents the number of the intervals under $q_2 + 1$ gridpoints.

B. AUGMECON2 with ILP

Our problem involves a tri-objective minimization. Using AUGMECON2, it is not necessary to use weights to represent the objective function. Instead, the objective function is:

$$min(WG), min(CN), min(CP). \tag{14}$$

To minimize WG, CN and CP with AUGMECON2, we set CN as the main objective function while the others as constraints, as solving CN consumes less time than solving the other two. The flow of AUGMECON2 integrated with ILP is shown in Fig. 9. It first builds the payoff table for all objectives, which includes the best and worst values for each objective. From the table, upper bounds and ranges can thus be obtained. Then the number of intervals divided in each range is set. In each interval, the ILP problem is solved minimizing the single objective function under the added constraints to obtain Pareto optimal solutions. The Pareto optimal solution generation terminates until all intervals have been solved or in any interval feasible solutions cannot be obtained. The flow ends with choosing the solution with the best performance.

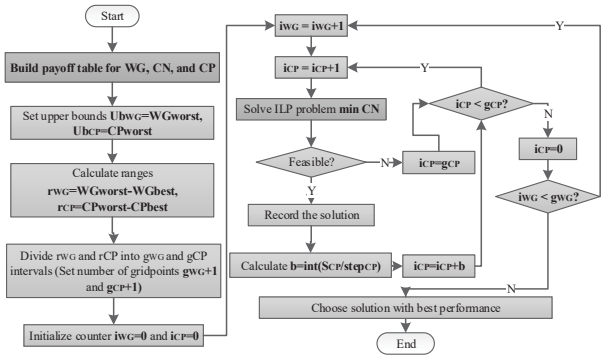


Fig. 9. Flow of AUGMECON2 for our problem.

The details of the payoff table and the ILP problem as shown in the dark gray blocks are introduced below.

1) Finding upper bounds and ranges. AUGMECON2 requires a payoff table consisting of the best and worst values of each objective, so the worst value is set as the upper bound for our minimization problem and the difference between the worst and the best value is the range. The best value is easily attained as the optimal value of the individual optimization of each objective. While the worst value is obtained through the lexicographic optimization [34] as follows.
   - Optimize CN and obtain $CN_{min} = z^*_{CN}$.
   - Optimize WG adding the constraint $CN = z^*_{CN}$, and obtain $WG_{min} = z^*_{WG}$.
   - Optimize CP adding the constraint $CN = z^*_{CN}$ and $WG = z^*_{WG}$, and obtain $CP_{min} = z^*_{CP}$.
2) Solve the ILP problem. The ILP problem is solved in each interval, so the ILP constraint in each interval is related to the corresponding gridpoint.
   - Objective: min($CN + eps \cdot (\frac{s_{CP}}{r_{CP}} + 0.1\frac{s_{WG}}{r_{WG}})$)
   - Constraints: besides the original constraints from (5) to (10), constraints $WG + s_{WG} = e_{WG}$ and $CP + s_{CP} = e_{CP}$ are added, where $e_{WG} = Ub_{WG} - (i_{WG} \cdot r_{WG})/g_{WG}$ and $e_{CP} = Ub_{CP} - (i_{CP} \cdot r_{CP})/g_{CP}$. s_{WG} and s_{CP} are non-negative surplus variables.

VII. Experiments

To show the efficiency of the proposed mapping approach, we exploit both synthetic and real-life benchmarks, including

task graphs from Standard Task Graphs (STG) [37], Task Graph For Free (TGFF) [38], SDF3 [39], and an executable H.264 baseline decoder. These applications are executed on a 2/4/8/16/24-CPU platform (denoted by 2/4/8/16/24P).

To compare the proposed approach with previous works on total execution time, the evaluation consists of three parts.

1) To test the effectiveness of applying communication pipeline in mapping, we compare the proposed ILP-based task mapping (denoted by **ILPM**) with [5] (denoted by ILP-NOCP) and [12](denoted by ILP-TACTC). Besides, as ILPM uses the new ILP model and the other two works use the old ILP model, we also compare the number of variables and solving time.
2) To test the advantage of the scalable part, we compare the proposed TP-CLUSTER algorithm (denoted by **TP-CLUSTER**) with ILPM, the heuristic solution for TACTC problem [12](denoted by ETA-TACTC), a scalable communication-aware task mapping for interconnected multicore systems [9](denoted by SCATM), and using SA to solve the problem of this paper(denoted by SA) on both performance and solving time.
3) To test the advantage of the adaptable part, we compare the proposed integration of ILP and AUGMECON2 (denoted by **ILP-AEC**) with its non-adaptive mapping ILPM for small-scale graphs or TP-CLUSTER for large-scale graphs, and the combination of weighted sum method (AWS) and ILP method (denoted by ILP-AWS) on both execution time and solving time.

The following subsections first introduce the benchmarks and the platforms, and then discusses the results.

A. Benchmarks

STG is a benchmark for evaluation of multiprocessor scheduling problems for DAGs. We exploit its actual application task graphs robot control with 88 tasks and 131 edges (denoted by ROBOT), sparse matrix solver with 96 tasks and 67 edges (denoted by SPARSE) and a part of fpppp with 334 tasks and 1145 edges (denoted by F4P) in the SPEC benchmarks. We randomly allocate communications to make each graph's CCR about 0.5. TGFF is a widely used DAG generator. We use TGFF to generate two task graphs with CCR about 0.5, one with 16 tasks and 22 edges (denoted by S-ACYC), and the other with 93 tasks and 138 edges (denoted by L-ACYC). Each application is executed for 10 times.

SDF3 is a synchronous dataflow graphs (SDFG) generator, and we use it to generate the cyclic task graphs corresponding to S-ACYC and L-ACYC with the same number of tasks and edges under similar CCR (denoted by S-CYC-E and L-CYC-E). To test ILP-ACE, we further create two cyclic task graphs with CCR less than 0.5 (denoted by S-CYC-S and L-CYC-S) with the same number of tasks and edges. Each application is executed for 10 times and the maximum delay number is 6.

H.264 baseline decoder is modeled as a cyclic task graph with 262 tasks and 680 edges (denoted by H264), and adopts a 300-frame CIF H.264 format Forman data stream as the input. Task execution time and communication size can be obtained from profiling before executed, and the CCR is 0.18.

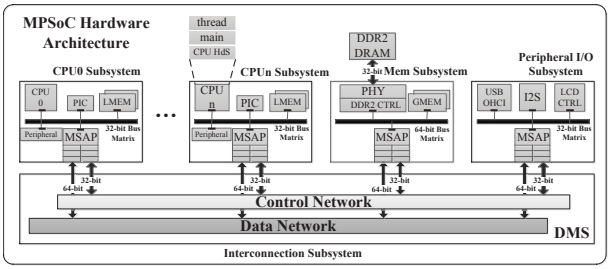


Fig. 10. Hardware architecture.

B. Platform

1) Hardware Platform: The whole approach is implemented on a CentOS 64-bit Linux Server with a 20-core Intel(R) Xeon(R) CPU E5-2670 at 2.70GHz and 96696MB RAM.

After getting the mapping result, we execute the real application on an MPSoC platform with flexible configurations and processor scalability as shown in Fig. 10. The platform contains at most 24 CPU subsystems, a memory subsystem, a peripheral subsystem and an interconnection subsystem. Each CPU subsystem uses a 32-bit local bus matrix to connect one processor with other local components. The processor type is configured as a 32-bit 7-stage pipeline CKCore RISC processor [40] without data cache. The Memory subsystem uses a 64-bit local bus matrix to connect on-chip global SRAM and off-chip DDR2 SDRAM. These three subsystems are connected with Distributed Memory Server (DMS) interconnection subsystems respectively through a Memory Service Access Point (MSAP) [41]. The DMS implemented by DMA acts as a server that provides the communication and synchronization services to the clients. Each MSAP delivers data transfer requests issued by its corresponding subsystem to other MSAPs via the control network.

2) Software platform: The ILP formulations are solved from IBM ILOG CPLEX 12.6.0.0. Other programs are implemented by C language, and compiled and linked by GCC.

The proposed mapping approach has been integrated with the Simulink-based MPSoC design platform, LESCEA multithreaded code generator [42]. LESCEA takes a Simulink-modeled application as an input, generates a set of multithreaded C codes and builds software stacks on targeting hardware architecture. As shown in Fig. 11, the general multithreaded code generation flow contains the following four main steps.

1) Task mapping. Tasks are bound to threads on processors, and one task is implemented by one thread. This paper mainly deals with this step.
2) Task scheduling. The execution sequence of tasks on each processor is determined after this step. This work uses the optimal scheduling policy to remove the scheduling interference. The discussion of scheduling is out of scope of this work.
3) Thread code generation. This step generates a set of C codes, including memory declarations and function calls related to the scheduling results, and maps the memory space and function parameters.
4) HdS adaption. This step generates main function codes for threads and initializes communication channels for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2859400, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

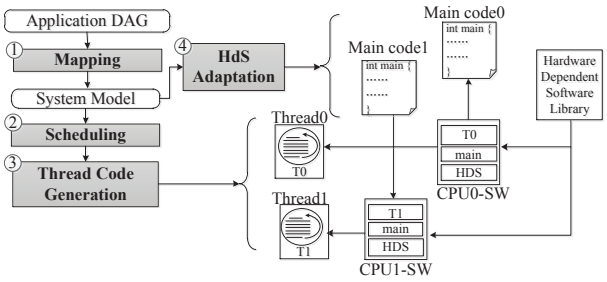


Fig. 11. LESCEA code generation flow.

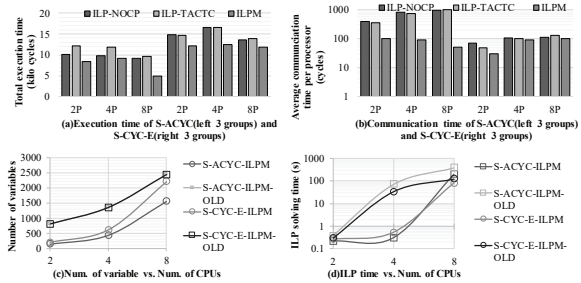


Fig. 12. Performance results of ILPM, ILP-NOCP, and ILP-TACTC.

CPU subsystem, as well as generates Makefile which links threads and HdS library.

C. Experimental Results

1) Comparison results for ILPM: ILP-NOCP is the processor mapping part in the ILP-based task mapping and scheduling method [5] considering load balance and inter-processor communication minimization. ILP-TACTC finds the optimal task assignment minimizing the system computation and communication cost under a given time constraint. We first compare ILPM with ILP-NOCP and ILP-TACTC to demonstrate the advantage of applying communication pipeline during mapping. As ILP-based methods can generate high-quality solutions fast for small-scale graphs, we present the results of small-scale graphs S-ACYC and S-CYC-E on 2/4/8P. Moreover, we set the objective weights of ILP-NOCP as 0.5 and ILPM as $\lambda_0 = \lambda_1 = 0.33$.

The experimental results are shown in Fig. 12. For both S-ACYC and S-CYC-E, ILPM shows better performance than ILP-NOCP and ILP-TACTC on 2/4/8P. From Fig. 12(a), compared to ILP-NOCP, the average improvement percentage is 23.04% for S-ACYC and 18.24% for S-CYC-E. Compared to ILP-TACTC, the average improvement is 33.94% for S-ACYC and 18.77% for S-CYC-E. The improvements are mainly contributed to the usage of communication pipeline. To further investigate the communication optimizations, we calculate the average communication time on each processor shown in Fig. 12(b). For S-ACYC, the communication time in ILP-NOCP and ILP-TACTC increases when the number of processors increases, while the communication time in ILPM decreases with the increasing number of processors. The result indicates that ILPM can effectively reduce communication cost even if the communication cost is exposed more with the increasing number of processors. However, although ILP-NOCP and ILP-TACTC try to minimize inter-processor communications, they cannot prevent the increase of communication cost as the number of processors increases. For S-CYC-E, although the communication cost increases with the increasing of processors due to cyclic topology, ILPM shows smaller communication cost than ILP-NOCP and ILP-TACTC.

From the communication pipeline mechanism, it can be used on all inter-processor communications for acyclic graphs. However, there is still communication cost for ILPM. The reason is that communication pipeline can only reduce communication transfer time, but in reality there are other sources for communication cost such as communication startup time and the communications out of the pipeline iterations. Therefore, even for acyclic graphs, communication cost cannot be totally hidden, and this work aims to reduce the major component of each communication, the communication transfer time.

We then compare ILPM and ILPM-OLD on solving time and the number of variables, as shown in Fig. 12(c) and (d). For both S-ACYC and S-CYC-E, ILPM consumes less time and has fewer variables than ILPM-OLD. The number of variables is decreased by an average of 79.13% for 2P, 61.09% for 4P, and 22.57% for 8P. Although the solving time of ILP does not only depend on the number of variables, reducing variables can reduce the solving time in a large degree.

The above results demonstrate the quality and solving time advantage of the proposed ILPM both on using communication pipeline and the new ILP model.

2) Comparison results for TP-CLUSTER: We compare TP-CLUSTER with ILPM, SCATM, ETA-TACTC and SA. SCATM is the scalable heuristic task mapping algorithm targeting at reducing communication cost. ETA-TACTC is the heuristic solution for the TACTC problem, and the solution involves a dynamic programming-based optimal tree assign algorithm. SA is a widely used meta-heuristic which can be used to solve the problem, and we tune its initial temperature, cooling ratio, frozen condition, and the number of iterations through experiments for different inputs. $15 \cdot 10^4$ is set as the acceptable number of iterations. We also set the ILP weights as $\lambda_0 = \lambda_1 = 0.33$, and ILP time limit as 1 hour. To show the scalability, we test large scale graphs L-ACYC, L-CYC-E, ROBOT, SPARSE, F4P, and H264 on 2/4/8/16/24P.

The experimental results are shown in Fig. 13. ILPM shows the best performance among all the comparison sets due to the optimal nature of ILP. However, for some large-scale cases such as F4P of 4/8/16/24P, we cannot obtain the ILP results within 1 hour, which demonstrates its scalability issue.

TP-CLUSTER can obtain results of all cases compared to ILPM, although it cannot achieve the quality as high as ILPM. The reason is that TP-CLUSTER tries to combine fine tasks into coarse tasks or divide the processors into clusters containing multiple processors, which in fact imposes constraints to the original ILPM model, and cannot reach the global optimum as ILPM. However, the average performance gap between TP-CLUSTER and ILPM is acceptable within 10% for acyclic graphs and 20% for cyclic graphs.

SCATM, ETA-TACTC, and SA can also obtain task mapping results. However, although SCATM and ETA-TACTC

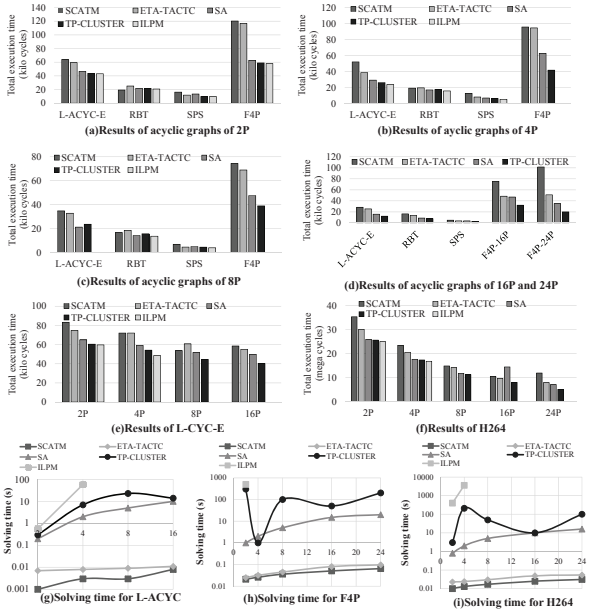


Fig. 13. Performance results of TP-CLUSTER and other methods.

try to minimize communication cost, they have not exploited communication pipeline to achieve further performance improvements. Compared to SCATM, TP-CLUSTER shows an average of 42.65% improvement for L-ACYC, 13.75% for ROBOT, 42.52% for SPARSE, and 58.59% for F4P, 25.15% for L-CYC-E and 32.00% for H264. Although SCATM spends the least time when giving the mapping solution, it has almost the worst solution quality due to its simple heuristic. Compared to ETA-TACTC, TP-CLUSTER shows an average of 34.74% improvement for L-ACYC, 20.84% for ROBOT, 16.94% for SPARSE, and 48.67% for F4P, 24.43% for L-CYC-E and 20.94% for H264. ETA-TACTC consumes larger solving time and gives better solutions than SCATM contributed to the dynamic programming method. As for SA, it gives better results than heuristic algorithms and consumes less time than TP-CLUSTER, but its solution quality cannot be guaranteed. For example, SA can achieve the best results such as ROBOT-8P or the worst results like H264-16P. Moreover, the result quality and convergence speed of SA highly depend on its parameters, which require delicate tuning. On the contrary, the proposed TP-CLUSTER can achieve a relatively high quality solution without complicated adjustments. Furthermore, with the increasing number of processors, the solving time of TP-CLUSTER may increase and decrease with the growing of the number of processors. The reason is that TP-CLUSTER may cluster only tasks or cluster both tasks and processors. In the former case, the solving time increases as the number of processors grows. While in the latter case, after a main small-scale ILP, the ILP in each cluster can run in parallel, resulting in much smaller solving time. Even in some cases, TP-CLUSTER spends similar solving time of SA.

The results show TP-CLUSTER can give fine results compared to other state-of-art heuristics within acceptable time.

3) Comparison results for ILP-AEC: We compare ILP-AEC with ILPM/TP-CLUSTER, and ILP-AWS. ILP-AWS is

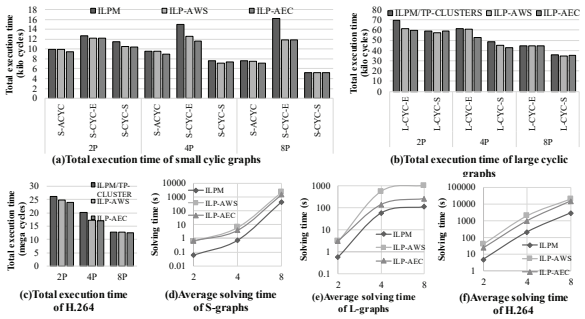


Fig. 14. Performance results of ILP-AEC and other methods.

the integration of AWS with ILP to solve the tri-objective problem where AWS [32] can adaptively adjust the weights to help obtain a set of Pareto optimal solutions. For acyclic graphs, CP weight has no impact on the ILP results, so for ILP-AWS, only WG and CN require weight adjustments. As the tradeoff between the three objectives is reflected in cyclic topologies, this part discusses only 1 acyclic L-ACYC and 5 cyclic S-CYC-E, S-CYC-S, L-CYC-E, L-CYC-S, and H264.

The experimental results are shown in Fig. 14. In all cases, the results of ILP-AEC and ILP-AWS perform no worse than ILPM/TP-CLUSTER, because both ILP-AEC and ILP-AWS can choose the result with best performance through a series of optimal mapping and communication pipeline allocation results for different computation and communication features. Further, ILP-AEC mostly has better results than ILP-AWS, because ILP-AEC can gradually explore the objective space, while ILP-AWS varies the weights which may lead to very close objective results. For the same number of points, ILP-AEC gives more efficient solutions. Compared to ILPM/TP-CLUSTER, the average improvement percentage of ILP-AEC is 5.56% for L-ACYC, 17.89% for S-CYC-E, 4.84% for S-CYC-S, 9.55% for L-CYC-E, 4.75% for L-CYC-S, and 6.50% for H264. Compared to ILP-AWS, the average improvement percentage of ILP-AEC is 4.94% for L-ACYC, 2.65% for S-CYC-E, 5.44% for L-CYC-E, and 0.45% for L-CYC-S. For H264, ILP-AEC achieves an average of 8.18% improvement compared to ILPM/TP-CLUSTER, and 1.77% compared to ILP-AWS, which shows the efficiency of ILP-AEC.

In some cases ILP-AWS and ILP-AEC have the same performance or even ILP-AWS performs better than ILP-AEC. As there is a coarse granularity of adjustment step for ILP-AWS and ILP-AEC, it is possible that each method finds parts of the points on the Pareto front and ILP-AWS shows better results than ILP-AEC. Moreover, although using the adaptive multi-objective method, it is also possible that ILP-AWS and ILP-AEC achieve the same result as ILPM/TP-CLUSTER as S-CYC-E on 2P and L-ACYC on 2/4P. The reason is that the predefined pair of weights of ILPM/TP-CLUSTER are just the optimal among all the weights and ILP-AWS, and the computation and communication just equal to the minimization of communication under the computation constraint for ILP-AEC.

We also plotted the solving time of all cases in Fig. 14(d)-(f). ILP-AEC consumes much less time than ILP-AWS, be-

cause ILP-AEC chooses CN as the objective consuming the least solving time, and ILP-AEC exploits the bypass coefficient of AUGMECON2 reducing the number of iterations. Moreover, the solving time for L-graphs (L-ACYC, L-CYC-E, L-CYC-S) of 8P is smaller than that of S-graphs (S-CYC-E, S-CYC-S). This is because the for L-graphs, ILPM cannot give results within the time limit, so TP-CLUSTER is exploited, and ILP of each processor cluster is performed in parallel. Therefore, the ILP scale and the solving time are both reduced.

The above results show the proposed approach can generate good results in acceptable time for different applications.

VIII. Conclusion

This paper proposes a novel ILP-based task mapping approach considering load balance and communication optimization. The approach consists of a new ILP mapping model which can efficiently reduce variables compared to previous ILP mapping model, and scalable and adaptable parts to tackle the scalability and adaptability issues of the ILP-based mapping. The effectiveness of the approach is evaluated on both synthetic and real-life benchmarks executing on a 24-CPU hardware platform. For the future work, communication pipeline can hide the communication transfer time to greatly reduce the communication cost, but its efficiency may be affected by applications with communication time larger than computation time, which requires future study. Moreover, besides task mapping, task scheduling also has great impact on performance and efficient scheduling policies are required. In addition, as heterogeneous MPSoC is becoming more popular, we will consider heterogeneity in future mapping methods.

References

[1] Y. H. Kang, W. J. Choi, B. C. Kim, and J. M. Kim, "On tradeoff between the two compromise factors in assigning tasks on a cluster computing," *Clust. Comput.*, vol. 17, no. 3, pp. 861–870, 2014.

[2] M. Q Sedeeq, M. Salih, O. F Yousif, and N. Q Mohammed, "Insight on MPSoC environment and its performance related to load balance," *ARPN J. Eng. Appl. Sci.*, vol. 12, 01 2017.

[3] J. Lee, M.-K. Chung, Y.-G. Cho, S. Ryu, J. H. Ahn, and K. Choi, "Mapping and scheduling of tasks and communications on many-core SoC under local memory constraint," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 11, pp. 1748–1761, Nov 2013.

[4] Y. Wang, D. Liu, Z. Qin, and Z. Shao, "Optimally removing intercore communication overhead for streaming applications on MPSoCs," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 336–350, 2013.

[5] K. Huang, M. Yu, X. Zhang, D. Zheng, S. Xiu, R. Yan, K. Huang, Z. Liu, and X. Yan, "ILP based multithreaded code generation for Simulink model," *IEICE Trans. Inf. Syst.*, vol. E97.D, no. 12, pp. 3072–3082, Dec. 2014.

[6] K. Huang, M. Yu, R. Yan, X. Zhang, X. Yan, L. Brisolara, A. A. Jerraya, and J. Feng, "Communication optimizations for multithreaded code generation from Simulink models," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 3, pp. 59:1–59:26, May 2015.

[7] Q. Tang, S. F. Wu, J. W. Shi, and J. B. Wei, "Optimization of duplication-based schedules on network-on-chip based multi-processor system-on-chips," *IEEE Trans. Parallel Distrib. Syst.*, no. 99, pp. 1–1, 2017.

[8] E. H. M. Cruz, M. Diener, L. L. Pilla, and P. O. A. Navaux, "An efficient algorithm for communication-based task mapping," in *Euromicro Intl. Conf. Parallel Distr. Network-Based Proces.*, 2015, pp. 207–214.

[9] I. H. Chung, C. R. Lee, J. Zhou, and Y. C. Chung, "Scalable communication-aware task mapping algorithms for interconnected multicore systems," in *IEEE Intl. Conf. High Perf. Comput. Comm.*, 2011, pp. 759–764.

[10] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Adaptive scheduling of task graphs with dynamic resilience," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 17–23, 2016.

[11] K. Huang, S. Xiu, M. Yu, X. Zhang, R. Yan, X. Yan, and Z. Liu, "Software pipeline-based partitioning method with trade-off between workload balance and communication optimization," *ETRI Journal*, vol. 37, no. 3, pp. 562–572, 2015.

[12] J. Liu, Q. Zhuge, S. Gu, J. Hu, G. Zhu, and H. M. Sha, "Minimizing system cost with efficient task assignment on heterogeneous multicore processors considering time constraint," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2101–2113, 2014.

[13] G. Mavrotas and K. Florios, "An improved version of the augmented ϵ-constraint method (AUGMECON2) for finding the exact pareto set in multi-objective integer programming problems," *Applied Math. Comput.*, vol. 219, no. 18, pp. 9652 – 9669, 2013.

[14] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous MPSoCs," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 1, pp. 14:1–14:25, Jan. 2015.

[15] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. 50th Annual Des. Autom. Conf.*, ser. DAC '13, 2013, pp. 1:1–1:10.

[16] H. Orsila, E. Salminen, and T. Hämäläinen, "Recommendations for using simulated annealing in task mapping," *Des. Autom. Embed. Syst.*, vol. 17, no. 1, pp. 53–85, 2013.

[17] A. Al-Wattar, S. Areibi, and G. Grewal, "Efficient mapping and allocation of execution units to task graphs using an evolutionary framework," *SIGARCH Comput. Archit. News*, vol. 43, no. 4, pp. 46–51, Apr. 2016.

[18] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 6, pp. 911–924, Jun. 2010.

[19] Q. Tang, T. Basten, M. Geilen, S. Stuijk, and J. B. Wei, "Mapping of synchronous dataflow graphs on MPSoCs based on parallelism enhancement," *J. Parallel Distrib. Comput.*, vol. 101, pp. 79–91, 2017.

[20] M. Lukasiewycz, M. StreubüHr, M. Glass, and C. Haubelt, "Combined system synthesis and communication architecture exploration for mpsocs," in *Des. Autom. Test Eur. Conf. Exhib.*, 2009, pp. 472–477.

[21] G. Karypis and V. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs," in *Proc. 1996 ACM/IEEE Conf. Supercomput.*, ser. Supercomputing '96, 1996.

[22] P. Sanders and C. Schulz, *Think Locally, Act Globally: Highly Balanced Graph Partitioning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 164–175.

[23] H. Wu, D. Lohmann, and W. Schröder-Preikschat, "A graph-partition-based scheduling policy for heterogeneous architectures," *CoRR*, vol. abs/1502.07451, 2015.

[24] S. G. Ahmad, C. S. Liew, M. M. Rafique, E. U. Munir, and S. U. Khan, "Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems," in *2014 IEEE 4th Intl. Conf. Big Data Cloud Comput.*, Dec 2014, pp. 129–136.

[25] H. P. Huynh, A. Hagiescu, W.-F. Wong, and R. S. M. Goh, "Scalable framework for mapping streaming applications onto multi-GPU systems," in *Proc. 17th ACM SIGPLAN symp. Princ. Pract. Parallel Progr.*, ser. PPoPP '12. New York, NY: ACM, feb 2012, pp. 1–10.

[26] S. Tosun, "Cluster-based application mapping method for network-on-chip," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 868 – 874, 2011.

[27] S. Kaushik, A. K. Singh, and T. Srikanthan, "Computation and communication aware run-time mapping for NoC-based mpsoc platforms," in *2011 IEEE Intl. SOC Conf.*, Sept 2011, pp. 185–190.

[28] S. Cotton, O. Maler, J. Legriel, and S. Saidi, "Multi-criteria optimization for mapping programs to multi-processors," in *2011 6th IEEE Intl. Symp. Ind. Embed. Syst.*, June 2011, pp. 9–17.

[29] S. E. Saidi, S. Cotard, K. Chaaban, and K. Marteil, "An ILP approach for mapping AUTOSAR runnables on multi-core architectures," in *Proc. 2015 Workshop Rapid Sim. Perf. Eval. Meth. Tools*, ser. RAPIDO'15. New York, NY, USA: ACM, 2015, pp. 6:1–6:8.

[30] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[31] K. Izui, T. Yamada, S. Nishiwaki, and K. Tanaka, "Multiobjective optimization using an aggregative gradient-based method," *Structural & Multidisciplinary Opt.*, vol. 51, no. 1, pp. 173–182, 2015.

[32] I. Y. Kim and O. L. D. Weck, "Adaptive weighted sum method for multiobjective optimization: a new method for pareto front generation," *Structural & Multidisciplinary Opt.*, vol. 31, no. 2, pp. 105–116, 2006.

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

[34] G. Mavrotas, "Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems," *Applied Math. Comput.*, vol. 213, no. 2, pp. 455 – 465, 2009.
[35] H. Amirian and R. Sahraeian, "Augmented ϵ-constraint method in multi-objective flowshop problem with past sequence set-up times and a modified learning effect," *Intl. J. Prod. Res.*, vol. 53, no. 19, pp. 1–15, 2015.
[36] R. Tarjan, "Depth-first search and linear graph algorithms," in *12th Annual Symp. Switching Automata Theor. (swat 1971)*, Oct 1971, pp. 114–121.
[37] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J. Sched.*, vol. 5, no. 5, pp. 379–394, 2002.
[38] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Hardw./Softw. Codes., 1998. (CODES/CASHE '98) Proc. Sixth Intl. Workshop*, Mar 1998, pp. 97–101.
[39] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Appl. Concur. Syst. Des., 6th Intl. Conf., ACSD 2006, Proc.*, June 2006, pp. 276–278.
[40] "C-SKY ip authorization," 2017, http://www.c-sky.com/solution/CPU-IP-shou-quan.htm.
[41] S.-I. Han, A. Baghdadi, M. Bonaciu, S.-I. Chae, and A. A. Jerraya, "An efficient scalable and flexible data transfer architecture for multiprocessor SoC with massive distributed memory," in *Proc. 41st Annual Des. Autom. Conf.*, ser. DAC '04, 2004, pp. 250–255.
[42] S.-I. Han, S.-I. Chae, L. Brisolara, L. Carro, R. Reis, X. Guérin, and A. A. Jerraya, "Memory-efficient multithreaded code generation from Simulink for heterogeneous MPSoC," *Des. Autom. Embed. Syst.*, vol. 11, no. 4, pp. 249–283, 2007.

Kai Huang received the Ph.D. degree in circuits and systems from Zhejiang University, China, in 2008. In 2006, he was a short-term Visitor with the TIMA Laboratory, Grenoble, France. From 2009 to 2011, he was a Post-Doctoral Research Assistant with the Institute of VLSI Design, Zhejiang University. In 2010, he also worked as a Collaborative Expert in VERIMAG Laboratory, Grenoble, France. Since 2012, he has been an Associate Professor with the College of Information Science and Electronic Engineering, Zhejiang University. His current research interests include embedded processors and SoC system-level design.

Xiaomeng Zhang received her B.S. degree from the College of Electrical Engineering, Zhejiang University, China, in 2013. She is currently pursuing the Ph.D. degree at the Institute of VLSI Design, Zhejiang University, Hangzhou, China. Her current research interests include multiprocessor software exploration and multithreaded code generation.

Dandan Zheng received the Ph.D. degree from Department of Information Science and Electronic Engineering, Zhejiang University, China, in 2009. She is currently a Research Assistant of College of Electrical Engineering at Zhejiang University. Her research interests are VLSI physical design of nano technology. Currently the main research is the low power physical design of MPSoC.

Min Yu received his B.S. and Ph.D. degrees from College of Information Science and Electronic Engineering, Zhejiang University, China, in 2009 and 2014, respectively. He is currently a research assistant in the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. His current research interests include performance estimation, high performance software exploration on multiprocessor and performance oriented automatic code generation on MPSoC.

Xiaowen Jiang received the B.S. degree from Jiangxi Normal University, Nanchang, China, in 2012. He is currently pursuing the Ph.D. degree at the Institute of VLSI Design, Zhejiang University, Hangzhou, China. His current research interests include multiprocessor software exploration, real-time systems and energy-efficient scheduling.

Xiaolang Yan received the B.S.E.E. and M.S.E.E. degrees from Zhejiang University, Hangzhou, China, in 1968 and 1981, respectively. From 1993 to 1994, he was a Visiting Scholar at Stanford University, Palo Alto, CA, USA. From 1994 to 1999, he was a Professor and the Dean of the Hangzhou Institute of Electronic Engineering, Hangzhou, China. Since 1999, he has been a Professor and the Director of the Institute of VLSI Design, Zhejiang University. His current research interests include embedded CPU design, SoC design methodology, and design for manufacturability.

Lisane B.de Brisolara is graduated from the Catholic University of Pelotas in Computer Science in 1999. She received the M.Sc and Dr. degrees in Computer Science from the Federal University of Rio Grande do Sul, UFRGS, Brazil, in 2002 and 2007, respectively. She is presently a professor at the Computer Science Department at the Federal University of Pelotas (UFPel), in charge of Software Engineering and Information Systems disciplines at the undergraduate levels. Her research interests include embedded systems modeling, design, validation, automation and test, and embedded software development.

Ahmed Amine Jerraya received the Engineering degree from the University of Tunis, Tunis, Tunisia, and two doctoral degrees in computer science from the University of Grenoble, Grenoble, France. He joined LETI, Grenoble, France, in 2007 as the Research Director and the Head of strategic design programs. Prior to joining LETI, he co-founded two startups and was the Research Director of the French National Center for Scientific Research (CNRS), Rennes, France. At CNRS, he also managed the TIMA laboratory's research dealing with MPSoC. He has co-authored eight books and published more than 200 papers in international conferences and journals.