Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Contention-Aware Mapping and Scheduling Optimization for NoC-Based MPSoCs

Rongjie Yan,[1,4] **Yupeng Zhou,**[2] **Anyu Cai,**[3] **Changwen Li,**[3] **Yige Yan,**[2] **Minghao Yin**[2,*]
[1]State Key Laboratory of Computer Science, ISCAS, Beijing, China
[2]Schlool of Computer Science and Information Technology, NENU, Changchun, China
[3]College of Computer Science, BJUT, Beijing, China
[4] University of Chinese Academy of Sciences, Beijing, China
yrj@ios.ac.cn, {zhouyp605, ymh}@nenu.edu.cn

Abstract

Network-on-Chip (NoC) has emerged as an alternative interconnecting paradigm in the state-of-the-art multi-core architectures. Its capability of parallel data transfer over various paths increases the possibility of resource contention and causes network congestion. How to avoid contention, as well as to optimize performance and energy consumption becomes a great challenge for system designers. In this paper, we consider spatial and temporal aspects of communication to avoid contention. In spatial aspect, we propose to utilize overlapped communication paths to estimate the degree of contention, such that they can be minimized in mapping stage. In temporal aspect, we sequentialize data transfer with potential contention by introducing additional latency. To optimize the design concerns with the corresponding constraint model, we further provide an efficient algorithm to search for better solutions for the problem, which integrates a local search process into a genetic algorithm and applies various heuristics in initialization and evolution process. Experimentations from random and real-case benchmarks demonstrate the efficiency of our method in multi-objective optimization and the effectiveness of our techniques in avoiding network contention, while keeping performance and energy consumption optimized.

Introduction

The scalable communication architecture makes network-on-chip (NoC) a prevailing solution for multiprocessor system-on-chips (MPSoCs). An important issue in the NoC paradigm is how to map and schedule tasks of an application to processing elements (PEs), such that the total energy consumption is minimized, and system performance can be optimized (Sahu and Chattopadhyay 2013). Within this architecture, communication is the main concern in the optimization process. To reduce the cost, tightly coupled tasks will be closely allocated, which may increase the possibility of communication *contention* for frequent data transfer over same paths. The increase in contention may incur long latency from network congestion thus leading to high energy consumption and poor system performance (Chou and Marculescu 2008; Yang et al. 2016).

*Corresponding author

With contention awareness, performance and energy consumption optimization needs to consider how to map tasks of an application to available PEs and tiles (the basic building block in a NoC which can accommodate one or more PEs), and how to schedule the execution of tasks on same PEs to avoid contention. However, path-based contention minimization that tries to minimize the possible contention on communication paths (Chou and Marculescu 2008) may reduce the degree of parallel execution between various tasks. Introducing additional latency to avoid overlapped communication in scheduling may degrade system performance. Therefore, reducing contention with less influence on performance and energy requires wary mapping and scheduling strategies, which is a challenge for designers.

In this paper, we consider time-triggered static scheduling for NoC-based MPSoCs, where both mapping and scheduling cannot be modified at runtime. During runtime, the generated application-specific scheduler acts as a look-up table, where every node in the platform should follow. System performance is evaluated by *makespan*, i.e., the scheduling length of the system. And energy consumption encompasses both computation and communication cost. To solve the performance and energy optimization problem with the consideration of potential contention in the constraint model, we integrate various heuristics and a problem-specific local search into a fast and elitist multi-objective evolutionary algorithm NSGAII (Deb et al. 2002).

The contributions of the paper are as follows. First, to reduce the over-approximation of spatial (path-based) contention in mapping, we propose to evaluate the contention degree in terms of overlapped communication paths, and regard it as an objective to be minimized. Temporal contentions can be avoided by introducing latency in involved communication during scheduling stage. Second, in the heterogeneous architecture, we consider a two-stage mapping to save makespan and communication cost: first mapping tasks to PEs, then mapping the PEs to tiles. This flexibility together with allowing more than one task per PE makes optimization even harder. Third, to deal with the optimization problem with multiple objectives, we construct a hybrid algorithm, by integrating a local search into an evolution process of NSGAII framework. The algorithm also

consists of problem-specific heuristics such as topology-oriented task clustering and capacity sensitive cluster refinement. Finally, we compare its performance with our implementation of NSGAII and exact solving techniques such as SMT (satisfiability modulo theories) solver Z3 (De Moura and Bjørner 2008), and MILP (mixed-integer linear and quadratic programming) as well as CP (constraint programming) solver CPLEX (CPLEX, IBM ILOG 2009). The experimental results show that the algorithm is capable of dealing with large-scale applications in approximating better Pareto fronts. The optimized solutions could save the effort of NoC-based design space exploration in looking for appropriate configurations and in estimating the performance in implementation. Though we limit our consideration to 2D mesh networks in this paper, our idea is applicable to other regular network topologies with static routing strategies.

Related work

Communication cost minimization is the main concern of NoCs (Sahu and Chattopadhyay 2013). To have a global view of the entire system, considering communication latency in task mapping (Chou and Marculescu 2008) and scheduling (Yang et al. 2016; Li and Wu 2016) is necessary for the design of NoCs. In contention-aware works, traffic congestions can be estimated using heuristics (He et al. 2012). The work in (Tino and Khan 2011) provides a contention model to analyze rendezvous interactions. Though the two works could be used to estimate the potential contention, they do not consider how to avoid it. Regarding every shared path between communication in mapping as a contention without temporal information (Chou and Marculescu 2008) is an over-approximation of contentions and may lead to a pessimistic performance estimation. The works in (Yang et al. 2016; Li and Wu 2016) sequentialize potential contentions in scheduling by considering priorities on communication, where the former mainly focuses on makespan minimization and the latter considers energy minimization. The strategy adopted in (Neubauer et al. 2018) blocks the whole route in transmission to avoid contention. We introduce a contention measure in terms of the overlapped paths and the distance of paths, and take it as a goal to be minimized in order to reduce the side-effect of contention over-approximation in mapping and introduced latency in scheduling. Compared with these works, our model considers spatial and temporal communication contention, and allows the overlapped communication without simultaneous link contention. Together with the optimization objectives, it can avoid contention, and reduce makespan.

In the content of static mapping and scheduling optimization (Sahu and Chattopadhyay 2013), where the deployment of applications to target platforms is calculated before the system is implemented, various algorithms exist, such as tabu search (TS) (Tino and Khan 2011), branch-and-bound (B&B) (Hu and Marculescu 2003), integer linear programming (ILP) (Chou and Marculescu 2008), population based incremental learning algorithm (PBIL) (Bolanos et al. 2013), genetic algorithms (GA) (Nedjah and de Macedo Mourelle 2014), simulated annealing algorithm (SA) (Chai et al. 2014), mixed ILP (MILP) (Yang et al. 2016), answer set programming modulo theories (ASPmT) (Neubauer et al. 2018). Due to the high complexity of the problem, methods with exhaustive search may not be efficient to provide solutions for large-scale cases. Most of the existing works employ meta-heuristic optimization. To deal with the contention-aware optimization of NoC-based MPSoCs, we further integrate a Pareto local search process into a multi-objective genetic algorithm, which is more efficient in dealing with large-scale applications.

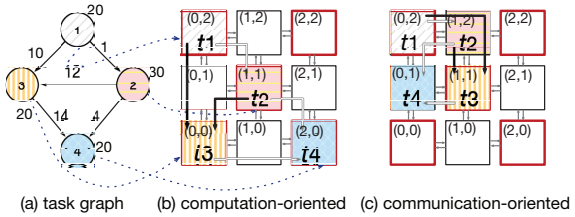


Figure 1: An application with various mapping paradigms.

The comparison between our work and aforementioned works is shown in Table 1, where `Heter.` and `Homo.` stand for heterogeneous and homogeneous architectures, `Map.`and `Sch.` stand for mapping and scheduling, respectively, and `Cap.` is for the capacity of PEs, i.e., the allowed number of tasks per PE. In the table, the first group manages to avoid contention, the second estimates the latency caused by contention without avoiding it, and the third does not consider contention, whose estimation may be too optimistic.

Preliminaries

We consider applications represented with task graphs. A task graph $G = \langle T, E \rangle$ is a directed acyclic graph, where T is a finite set of tasks and $E \subseteq T \times T$ is a set of precedence relations between tasks. Each task $t \in T$ is associated with the amount of work to be dealt with, and each edge $e_{ij} \in E$ is associated with the amount of data transferred between tasks t_i and t_j. For example, the model shown in Fig.1(a) is a task graph with 4 tasks and 5 dependency relations.

The target hardware is a 2D mesh NoC-based heterogeneous MPSoCs. Each tile z contains a PE and is connected to a *router*. Routers are connected with each other through bidirectional *links*. XY-routing is assumed in this paper, i.e., X direction is determined first and then Y direction is determined. Data transmission is in the unit of packets. If two consequential tasks are mapped to the same PE, data can be directly read without routing. Within this topology, the location of each PE can be determined by the position of accommodated tile z, i.e., a pair of coordinates (x, y). And the distance between any two tiles at (x_k, y_k) and $(x_{k'}, y_{k'})$ is:

$$\beta_{kk'} = abs(x_k - x_{k'}) + abs(y_k - y_{k'}) \quad (1)$$

We present a 3×3 NoC with two different types of PEs in Fig.1(b), where the frequency of PEs located at the tiles with bold red square is twice as fast as others. To compare the computation and communication consumption over various configurations of an application, we provide two mappings for the model in Fig.1(a), where one utilizes faster PEs and

Table 1: Comparison of related works

Reference	Arch.	Model	Objective	Cap.	Alg.
Our work	Heter.	Map.+Sch.	Makespan+Energy+Contention	Multiple	Hybrid
(Chou and Marculescu 2008)	Homo.	Map.	Communication+Contention	Single	LP Approx.
(Yang et al. 2016)	Homo.	Map.+Sch.	Makespan	Multiple	Heuristic
(Li and Wu 2016)	Homo.	Map.+Sch.	Energy	Single	Stepwised
(Neubauer et al. 2018)	Heter.	Map. +Sch.	Latency+Energy+Area	Multiple	ASPmT
(He et al. 2012)	Heter.	Map.+Sch.	Makespan+Energy	Multiple	MILP
(Tino and Khan 2011)	Homo.	Map.	Power+Throughput	Single	TS
(Hu and Marculescu 2003)	Homo.	Map.	Communication	Single	B&B
(Nedjah and de Macedo Mourelle 2014)	Heter.	Map.	Area+Makespan+Power	Multiple	GA
(Chai et al. 2014)	Homo.	Sch.	Makespan+Load balance	Multiple	SA
(Bolanos et al. 2013)	Heter.	Map.+Sch.	Reliability	Multiple	PBIL

Table 2: Notation explanation for constants and variables

Notation	Explanation
τ (ε)	time (energy) for transferring a packet through a link
τ' (ε')	time (energy) for routing a packet through a router
μ	bandwidth of a link
ξ	time bound for simultaneous communication
ρ_k	processing speed of PE p_k
ω_k	capacity limit for p_k
$\beta_{kk'}$	Manhattan distance between tiles z_k and $z_{k'}$
α_i	computation volume for task t_i
ζ_{ij}	communication volume between tasks t_i to t_j with $e_{ij} \in E$
δ_{ij}	true when there is a dependency relation between tasks t_j and t_i
γ_{ijlr}	number of shared links between XY-routing paths from tile z_i to z_j and from z_l to z_r
m_{ik}	true for task t_i being mapped to p_k
ℓ_{ik}	true for task t_i being located at tile z_k
s_i, f_i	start and end time of executing task t_i
$\hat{s}_{ij}$, $\hat{f}_{ij}$	start and end time of data transfer for e_{ij}
$q_{kk'}$	true if PE p_k is allocated to tile $z_{k'}$
n_{ij}	true for task t_j being executed next to t_i on the same PE
o_{ij}	true if task t_j depends on t_i and they are not in the same PE
d_{ij}	communication distance for communication from task t_i to t_j
c_{ijlr}	true for existing conflicts between data transfer e_{ij} and e_{lr}

the other minimizes communication. We only consider the cost in links, and ignore the latency caused by routing and contention. The communication is assumed to have cost 1 per packet payload per link, to have an intuitive comparison between various configurations. If we do not consider the additional waiting cost for data acquisition, and savings for parallel data transfer, in the mapping of Fig.1(b), the time costs of computation and communication are 45 and 82, respectively. For the case in Fig.1(c), the time costs are 70 and 55, respectively. The two mappings both have a contention between communication paths from t_1 to t_3 and from t_2 to t_3, marked with bold lines. Meanwhile, a contention exists between paths from t_1 to t_3 and from t_1 to t_2 in Fig.1(c), which does not exist in the case of Fig.1(b).

Constraint formulation for NoC-based design

Given an application with $G = \langle T, E \rangle$, a NoC with $n \times n$ tiles and a set of PEs P with $|P| = n^2$, we present the notations used in the model in Table 2, where the constants in the first part of the table can be extracted according to application and platform features, or be calculated according to routing algorithms (e.g. number of shared links between two paths). We list the decision variables and intermediate variables in the second and third parts of Table 2.

Constraint modeling

Our constraint model has the following features: First, the mapping of PEs and tiles is not fixed. The mapping consists of two stages: tasks to PEs and PEs to tiles. Second, multiple tasks can be mapped to one PE, which makes scheduling more complicated. Third, the constraints are in the format of logical formulas, which can be translated to the formats of various solvers. The constraints can be divided into two types: static constraints and dynamic behaviors. Static constraints encode the mapping and communication relation between tasks, PEs and tiles. Behavior constraints decide the scheduling sequence of computation and communication.

To simplify the representation, we introduce notations $\mathbb{T} = \{1, \ldots, |T|\}$ and $\mathbb{P} = \{1, \ldots, |P|\}$ to denote the range of tasks and PEs, respectively.

Static constraints

$$\begin{aligned} &\forall i \in \mathbb{T}, \forall k \in \mathbb{P}, m_{ik} \rightarrow \bigwedge_{k' \neq k} \neg m_{ik'}, \ell_{ik} \rightarrow \bigwedge_{k' \neq k} \neg \ell_{ik'} \\ &\forall i \in \mathbb{T}, \exists k, k' \in \mathbb{P}, m_{ik} \wedge \ell_{ik'}, m_{ik} \wedge \ell_{ik'} \rightarrow q_{kk'} \end{aligned} \tag{2}$$

$$\forall k \in \mathbb{P}, (\sum_{i=1}^{|T|} m_{ik}) \leq \omega_k \tag{3}$$

$$\forall i, j, l \in \mathbb{T}, \exists k \in \mathbb{P}, m_{ik} \wedge m_{jk} \wedge m_{lk} \wedge n_{ij} \wedge n_{jl} \rightarrow \neg n_{il} \tag{4}$$

$$\forall i, j \in \mathbb{T}, \exists k, k' \in \mathbb{P}, \delta_{ij} \wedge m_{ik} \wedge m_{jk'} \wedge (k \neq k') \rightarrow o_{ij}, o_{ij} \rightarrow \delta_{ij} \tag{5}$$

$$\begin{aligned} &\forall k_1, k_2, k_3, k_4 \in \mathbb{P}, \exists i, j, l, r \in \mathbb{T}, (\gamma_{k_1 k_2 k_3 k_4} > 0) \wedge \\ &\ell_{ik_1} \wedge \ell_{jk_2} \wedge \ell_{lk_3} \wedge \ell_{rk_4} \wedge o_{ij} \wedge o_{lr} \rightarrow c_{ijlr} \end{aligned} \tag{6}$$

$$\forall i, j \in \mathbb{T}, \exists k, k' \in \mathbb{P}, o_{ij} \wedge \ell_{ik} \wedge \ell_{jk'} \rightarrow (d_{ij} = \beta_{kk'}) \tag{7}$$

The two-stage mapping of task-to-PE and PE-to-tile could be converted to the mapping between task and PE, and the mapping between task and tile. That is, when a task is mapped to a PE and a tile respectively, the relation between the PE and the tile is fixed. Constraint 2 requires that one task can only be mapped to one PE and one tile, respectively. Constraint 3 regulates that the number of mapped tasks for a PE cannot violate the capacity limit. Constraint 4 explains that the stepwise execution relation is not transitive. Constraint 5 reasons the case with communication relation. That is, communication exists when two tasks with dependency relation are allocated to various PEs. Constraint 6 shows the case of potential communication contention. Constraint 7 explains how to calculate the communication distance between two dependent tasks, which is decided by the distance between the mapped tiles.

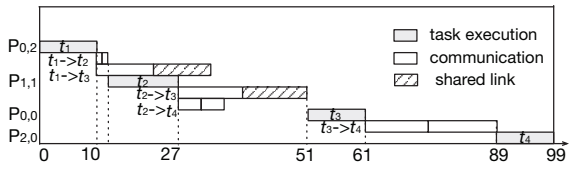


Figure 2: A schedule for the mapping in Fig.1(b).

Behavior constraints

$$\forall i \in \mathbb{T}, f_i = s_i + \alpha_i/(\textstyle\sum_{k=1}^{|P|} m_{ik} \cdot \rho_k) \tag{8}$$

$$\forall i,j \in \mathbb{T}, \hat{f}_{ij} \geq \hat{s}_{ij} + o_{ij} \cdot (\zeta_{ij} \cdot \tau \cdot d_{ij}/\mu + \tau' \cdot (d_{ij}+1)) \tag{9}$$

Constraints 8 and 9 present quantitative relation for task execution and data transfer, respectively. The cost of data transfer depends on the distance of communication, and the costs of links and routers. The path involving d_{ij} links will go through $d_{ij}+1$ routers, where the first packet of a message needs to be routed, and the rest can follow the first.

$$\forall i,j \in \mathbb{T}, \delta_{ij} \rightarrow \hat{f}_{ij} \leq s_j, \quad \text{and } f_i \leq \hat{s}_{ij} \tag{10}$$

$$\forall i,j \in \mathbb{T}, n_{ij} \rightarrow f_i \leq s_j \tag{11}$$

Constraint 10 regulates the causality between task execution and communication constrained by the dependency relation. Constraint 11 requires that for two tasks scheduled next to each other in the same PE, the start of the latter should wait for the finish of the former.

$$\forall i,j \in \mathbb{T}, \exists k \in \mathbb{P}, m_{ik} \wedge m_{jk} \rightarrow s_j \geq f_i \vee s_i \geq f_j \tag{12}$$

$$\forall i,j,l,r \in \mathbb{T}, \exists k \in \mathbb{P}, m_{ik} \wedge m_{lk} \rightarrow \hat{s}_{ij} \geq \hat{f}_{lr} \vee \hat{s}_{lr} \geq \hat{f}_{ij} \tag{13}$$

$$\forall i,j,l,r \in \mathbb{T}, c_{ijlr} \wedge (abs(f_i - f_l) \leq \xi) \rightarrow \hat{s}_{ij} \geq \hat{f}_{lr} \vee \hat{s}_{lr} \geq \hat{f}_{ij} \tag{14}$$

Constraints 12 and 13 specify the non-overlapping relation for task execution on the same PE and communication from the same source, respectively. Constraint 14 says that two communication paths with shared resource cannot overlap, if their sources finish execution almost simultaneously. This constraint aims to allow the case with path but without temporal communication contention. It covers path contention with same sources or destinations. Reconsider the mapping in Fig.1(b). The communication path from t_1 to t_3 and the path from t_2 to t_3 have a shared link. But the two do not request the shared link at the same time, where the communication of the first can be finished before that of the second arrives, as shown in Fig.2.

Optimization objectives

We evaluate the makespan with the scheduling length of the task graph, i.e., the point when the last task terminates:

$$\mathcal{M} = \max\{f_i \mid t_i \in T\} \tag{15}$$

Energy consumption includes cost for computation and communication. For computation, we distinguish dynamic and static status in a PE and accumulate the total cost. For communication, we consider the cost for routing and data transfer. In a NoC network, some PEs may not have any allocated tasks. We use $P' = \{p_k \in P \mid \sum_{i=1}^{|T|} m_{ik} > 0\}$ to denote the set of occupied PEs.

Let $\mathcal{E}_{d_k}$, and $\mathcal{E}_{i_k}$ be the rates of dynamic, static (idle) power consumption of PE p_k, respectively. Let $\mathcal{L}_k = \sum_{i=1}^{|T|} m_{ik} \cdot \alpha_i/\rho_k$ be the load of computation on p_k . The computation cost (E_p) is:

$$E_p = \sum_{p_k \in P'} (\mathcal{E}_{d_k} \cdot \mathcal{L}_k + \mathcal{E}_{i_k} \cdot (\mathcal{M} - \mathcal{L}_k)) \tag{16}$$

The energy cost for communication E_m is evaluated w.r.t. the amount of data and the distance of the transmission.

$$E_m = \sum_{p_k, p_{k'} \in P'} \sum_{i,j=1}^{|T|} \zeta_{ij} \cdot o_{ij} \cdot m_{ik} \cdot m_{jk'} \cdot (\varepsilon \cdot \beta_{kk'} + \varepsilon' \cdot (\beta_{kk'} + 1)) \tag{17}$$

In addition to energy and makespan optimization, we also expect to reduce potential contention, such that the effort in scheduling is reduced. We adopt overlapped path between two communication paths to measure the degree of potential contention. Suppose two paths are from tile z_{k_1} to z_{k_2}, and z_{k_3} to z_{k_4}, respectively. Their contention degree is

$$\wp_c(k_1,k_2,k_3,k_4) = \gamma_{k_1k_2k_3k_4}/(\beta_{k_1k_2} \cdot \beta_{k_3k_4}) \tag{18}$$

Let $\mathbb{S}_i$ be the set of indexes for successors of task t_i. The total contention degree between two tasks t_{i_1} and t_{i_2} is

$$\mathcal{P}_{i_1i_2} = \sum_{j_1 \in \mathbb{S}_{i_1}, j_2 \in \mathbb{S}_{i_2}} o_{i_1j_1} \cdot o_{i_2j_2} \cdot \wp_c(k_{i_1}, k_{j_1}, k_{i_2}, k_{j_2}) \tag{19}$$

To avoid high local contention degree, we apply the average contention degree to evaluate the quality of the allocation strategy with Equation 20.

$$\bar{\mathcal{P}}_c = \sum_{i,j=1}^{|T|} abs(\mathcal{P}_{ij} - \sum_{i,j=1}^{|T|} \mathcal{P}_{ij}/|T|) \tag{20}$$

Altogether, we have three objectives to be optimized: makespan, energy cost, and contention:

$$minimize(\mathcal{M}),\ minimize(E_p + E_m) \tag{21}$$

$$minimize(\bar{\mathcal{P}}_c) \tag{22}$$

Discussion. The number of constraints is polynomial to the number of tasks and PEs. In the constraint formulation, Constraint 14 and Objective 22 are complementary. Constraint 14 considers the temporal relation of contention, and separates the potential overlapped communication to avoid contention. Objective 22 focuses on spatial contention and tries to minimize the overlapped communication paths.

The hybrid search algorithm

The complexity of the constraint formulation and the feature of multiple objectives make the optimization process more difficult. Searching for exact solutions may be infeasible for large applications. Accordingly, we integrate a Pareto local search method into the NSGAII framework together with objective-related heuristic extensions. The resulting multi-objective optimization with hybrid search (MOHA) consists of three optimization stages. The first is a task-classification process, including task clustering (TC) and cluster refinement (CR), to group tightly coupled tasks into one PE, where communication effort and the degree of path contention can

Algorithm 1 TC(G)

```
1: Input: task graph G
2: Output: Scheduled queues C
3: q = 0; k = 0; i = 0;
4: for ∃t.Pre(t) = ∅ do
5:     add t to Q(q); q++;
6: while (Q(k) ≠ ∅) ∧ (k < q) do
7:     t = pop(Q(k)); C(i).push(t);
8:     for t' ∈ Succ(t) do
9:         t'.in − −;
10:    if {t'|t' ∈ Succ(t) ∧ t'.in == 0} == ∅ then
11:        i++;
12:        if Q(k) == ∅ then
13:            k++;
14:    else
15:        for t' ∈ {t'|t' ∈ Succ(t) ∧ t'.in == 0} do
16:            Q(k).push(t');
17: return C;
```

be reduced. Then, with the heuristic of spiral mapping (SM), a PE type and PE-to-tile location mapping stage is applied to optimize makespan and average contention degree. The two stages are integrated into initialization. In the evolution process, with the integrated local search and the CR heuristics, we look for a better task-to-PE mapping and scheduling sequence for tasks in every PE, to minimize the makespan.

Optimization-related heuristics

Task clustering (TC) To reduce communication cost, tightly coupled tasks should be allocated together. Therefore, a task graph is explored according to dependency relation to divide the tasks into different clusters. Let *Pre*(t), *Succ*(t) be the sets of predecessors and successors of t, respectively, and $t.in$ be the number of unscheduled predecessors of t, which is initialized as $|$*Pre*$(t)|$, the number of its predecessors. In Alg.1, we use two queues Q and $\mathcal{C}$ to record the tasks to be organized and having been scheduled, respectively. Starting from a task t with *Pre*$(t) = \emptyset$, the heuristic pushes all its *schedulable* successors t' (where all the tasks in *Pre*(t') have been pushed into $\mathcal{C}$) into the same cluster in which t is, and other tasks in $\{t'|t' \in$ *Succ*$(t) \wedge t'.in \neq 0\}$ into the cluster in which its last being scheduled predecessor is, to maximize computation parallelization.

Reconsider the model shown in Fig.1(a). As *Pre*$(t_1) = \emptyset$, initially Q contains one queue with t_1 (Alg.1 Line 5). In the `while` loop, t_1 is popped and added to the first cluster in $\mathcal{C}$ (Alg.1 Line 7). Then $t_2.in$ and $t_3.in$ are decreased by 1 (Alg.1 Line 9). Next t_2 can be added to $Q(0)$ (Alg.1 Line 16) and be processed in the second loop. Finally, the four tasks are added into the same cluster.

Capacity constrained cluster refinement (CR) We assume that the sum of capacity limits of all PEs is larger than or equal to the number of tasks, to ensure that the system is schedulable. However, after TC, the number of clusters may go beyond the number of available PEs, or the number of involved tasks in a cluster may exceed the capacity of a PE. In the first case, the clusters are ranked in a descending order of computation load. Each time the last two clusters are merged and the clusters are re-ordered, until the number of clusters meets the requirement. In the combination, the ordering of tasks in one cluster is unmodified, and the tasks in the other are inserted sequentially according to the scheduling order of their predecessors. In the second case, we can apply precaution or remedy strategies. The first strategy can generate a feasible solution directly, and be integrated into clustering stage or local search stage. In this strategy, a task will be allocated to a PE in which the number of accommodated tasks is below the capacity limit. The second strategy is to deal with an infeasible solution generated from a random mode, such as random initialization or evolution process. For a cluster in an overloaded PE, let t_i be the last scheduled task of the cluster. We search for its successor t_j and check the availability of PE that accommodates t_j. If such a PE exists, we insert t_i in front of t_j. Otherwise, we append t_i to the cluster of an available PE randomly. In such a case, indirect dependency between the existing tasks and t_i may be violated. Let $\mathcal{T}$ be the set of tasks in the cluster before modification. We execute a breadth-first search from t_i in the task graph. If a task t_j in $\mathcal{T}$ is explored, the indirect dependency is violated, and we insert t_i in front of t_j. Otherwise, the new generated cluster is feasible. The loop continues until the capacity constraint is satisfied.

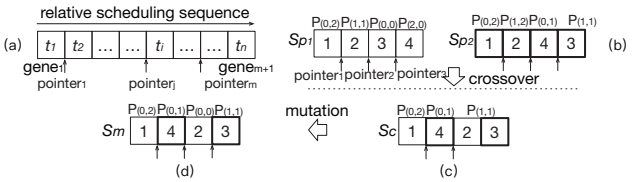


Figure 3: Chromosome encoding and genetic operations

Spiral mapping (SM) We first calculate communication cost between all clusters and total computation cost of every cluster. The allocation of a cluster is searched in a spiral path (Mehran et al. 2007) from centre to the boundary of the network architecture, such that communicating clusters are close to each other. Then, the cluster with the highest amount of work is placed to the centre with the fastest PE, and the one with the highest transmission cost to the allocated cluster is placed next to it. If there are several feasible clusters, we select one randomly.

MOHA algorithm

The essential ingredients of genetic process in MOHA consist of the problem-oriented chromosome encoding, the initialization, the operations to generate offsprings, and how local search is integrated into.

Encoding and decoding To integrate the mapping and local scheduling strategy, a gene is an identity of PE together with a sequence of identities for the scheduled order of allocated tasks. In a chromosome, only the occupied PEs are recorded, as the mapping between PE and tile is fixed in

initialization, and the genetic process will not modify the relation. And the indexes of PEs may not be continuous but should be ordered w.r.t. their positions. When tasks are allocated to unused PEs in the evolution process, the indexes of the PEs will be inserted into the chromosome with the indexes of the accommodated tasks. We need extra pointers to separate the genes for occupied PEs in a chromosome, as shown in Fig.3 (a). For the example in Fig.1(b), its encoding is shown in S_{P1} of Fig.3(b). As four PEs are occupied in S_{P1}, we need three pointers to separate the genes.

The decoding of a chromosome is to extract physical locations of tasks w.r.t. the mapping information, the computation cost of every task w.r.t. the allocated PE, and the task scheduling sequence within a PE.

Initialization A diversified population could provide a greater chance to find the optimal, yet would result in slow convergence. While an intensified population could converge fast, yet would make the algorithm trap into local optimum. Therefore, we randomly generate some candidates to cover a more comprehensive solution domain, and construct a solution according to *TC*, *CR* and *SM* heuristics for the reduction of communication cost and a faster convergence. Then they will be divided into various sub-populations by the fast non-dominated sorting and crowding distance calculation in NSGAII. Though only one solution is generated by the heuristics, information obtained from heuristics can be partially preserved in their offspring.

Randomly generated solutions in initialization and those generated in the genetic process may not be feasible. We use a *repair* operator to repeatedly check the feasibility of solutions and ensure that each task starts after all its predecessors have been executed. That is, the operator checks whether direct or indirect dependency exists and the causality relation is maintained from the beginning of a gene. Once a violation is detected, the task that should be scheduled earlier is inserted before its direct or indirect successor. The process terminates when no violation exists.

Genetic process We propose a uniform crossover and a split (or merge) based mutation operator in the genetic procedure, to improve the feasibility and diversity of solutions. The crossover operator is to bring the search process to new promising regions to diversify the solutions. It performs after a random selection of two parents from the dynamic elitist population and the evolution population. Each task in the offspring inherits the position from one of its parents randomly. For example, two parents S_{P1} and S_{P2} are shown in Fig.3 (b). Their child S_c in Fig.3(c) is generated from crossover operator, where tasks t_3 and t_4 inherit the mapping from S_{P2}, and the others inherit from those of S_{P1}.

To avoid that the integrated local search on a fixed mapping is trapped into a local optimum, each individual is evaluated with a mutation probability. In the mutation operator, two operations can be selected with probability $\mathcal{P}_o = |P'|/|P|$ (the ratio of occupied PEs). That is, when PE utilization is low, we split clusters. Otherwise, we merge clusters. In the split operator, a cluster with the highest computation cost is decomposed into two with the balanced load. For example, the ratio of occupied PEs in the generated child S_c of Fig.3(c) is low. In the mutation, the cluster in the last PE containing two tasks is split into two and task t_2 is allocated to an available faster PE, as shown in Fig.3(d). In the merge operator, two clusters with the lowest computation costs are selected and merged. The new generated clusters from split or merge operations will be allocated to the first available PEs according to the SM heuristic. And CR heuristic ensures that the number of tasks in every PE is under the capacity constraint.

Pareto local search (LS) Based on solutions obtained from the crossover and mutation operations, we apply a Pareto local search at the end of a genetic step to enhance the intensive searching capability. The key of the procedure is an insertion-based neighborhood comparison. Given a candidate solution *S* and tasks t and t', if we insert task t in front of task t' and keep relative positions of other tasks unchanged, we can obtain a neighbor solution of *S*. We use *neighbor(S_t)* to denote the set of neighbors of *S* by inserting t into other positions, without destroying the precedence relation of the rest of tasks to save the repairing effort. In the main Pareto local search loop, a task t of solution S is randomly selected, and checked to find a solution that can dominate S from its neighborhood space. Specifically, if *neighbor(S_t)* is better than *S* based on Pareto optimal evaluation, it will replace *S* for further exploitation and the iteration is restarted to look for a better neighborhood. Meanwhile, the elitist population is updated. Otherwise, another task will be checked and the iteration will be continued. This process continues until the times of iteration reach the maximum limit.

Experimentation

MOHA together with NSGAII is implemented in C++. The experimentation runs on a PC with intel 2.2 GHz processor and 8.0 GB memory. The effectiveness of the model and the efficiency of MOHA are evaluated with randomly generated cases from TGFF tool (Dick, Rhodes, and Wolf 1998) and a set of real benchmarks: H264 and MP3 decoders, TMNR procedure, Multi-Window Display (MWD) application, Livermore Loop (FFT_loop), and Bufferfly (FFT_butterfly).

Experimental setup

The setting of parameters for computation and communication cost in the constraint model can be found in (Yang et al. 2016). We assume that two types of PEs with various frequencies but the same capacity are available. Except for the fine-grained H264 case that is deployed in various scales of platforms, all other experimentations take a 2×2 2D mesh NoC as the target platform. We generate small and large-scale random cases. To check the sensitivity of various methods to computation load, the volume of computation from every case has been multiplied three times (suffixed with "-m" for the original, and "-p" for after multiplication).

We adopt our implementation of NSGAII as the baseline without any heuristics. Both MOHA and NSGAII share the same parameter setting to reach a fair comparison: the size of population is 30, and the probability of mutation operation is 0.1. They are executed 5 times on each instance with a time limit of 1000 seconds, and the average value is recorded. For

Table 3: Comparison between exact and heuristic methods

Case	$\|T\|$	$\|E\|$	MOHA	NSGAII	MILP		CP		Z3	
			num.	num.	num.	time	num.	time	num.	time
5-m	5	4	3	2(=)+2(≻)	2(=)	82.57	3(=)	13.35	3(=)	3.68
5-p	5	4	3	3(=)	2(=)	106.20	3(=)	17.75	3(=)	6.49
7-m	7	6	2	2(=)	2(=)	114.20	2(=)	46.14	2(=)	9.14
7-p	7	6	2	2(=)	2(=)	154.50	2(=)	75.16	2(=)	19.85
8-m	8	7	2	2(=)	2(=)	497.65	2(=)	59.70	2(=)	12.92
8-p	8	7	2	2(=)	2(=)	323.48	2(=)	130.87	2(=)	23.13
10-m	10	9	1	1(=)	1(=)	2620.71	1(=)	-	1(=)	93.45

fine-grained H264 case, the time limit is set to 3600 seconds. The iteration limit of local search in MOHA is set to 5.

Evaluation on MOHA

We compare the quality of solutions between MOHA, NSGAII, and exact methods for small-scale cases with makespan and energy optimization, and the effectiveness of various heuristics in MOHA with NSGAII for large-scale and real cases with three objectives.

Comparison on the quality of solutions The comparison with exact solving techniques involves popular tools Z3 (supporting multi-objective optimization and providing all Pareto fronts) for SMT solver, CPLEX for MILP and CP solvers (called iteratively to search for better solutions and obtain the Pareto fronts). We construct various formats of models[1] w.r.t. the proposed constraint model and the features of the corresponding solvers, and build the procedures to search for Pareto fronts with MILP and CP solvers.

When given two optimization objectives, CPLEX (for both MILP and CP) cannot finish optimization within a time limit (3600 seconds) for case 10-p. When given three objectives, the MILP solver fails to provide any result within several days, and Z3 can only provide solutions for TMNR, MWD and MP3 decoder within 10 hours. Hence, we compare MOHA with CPLEX and Z3 using small-scale cases for makespan and energy minimization.

In Table 3, the time cost above 3600 seconds is denoted by "-", the legend *num.* in MOHA shows the number of generated non-dominated solutions. The other columns with legend *num.* also show the relation between the generated solutions with MOHA, where $\succ$ means that solutions of NSGAII are dominated by those of MOHA, and $=$ means that solutions of the methods are the same.

In the experimentation, MOHA can find all the Pareto fronts. The performances of MOHA and NSGAII are similar. However, in the first case, though NSGAII can find four solutions, two of them are dominated by those of MOHA. And the other two are in MOHA. Z3 can always find all the solutions quickly, for the constraints can be encoded as logic formulas and the optimization process is fast for small-scale cases. Compared with the MILP solver, the better performance of CP solver is achieved from the logic constraint reasoning and the scheduling-oriented features. The performance of the three exact methods decreases with the increasing number of tasks and the increasing load of computation.

[1]Available from https://github.com/LIIHWF/ICAPS2020

Heuristics comparison with various metrics To check the effectiveness of the heuristics, we present the comparison with NSGAII from Set Coverage (Zitzler et al. 2003) to reflect the percentage of dominance measured between different heuristics and algorithms, and hyper volume indicator I_H (Zitzler and Thiele 1999) to reflect both dominance degree and distribution of Pareto fronts for various algorithms:

- Let *A* and *B* be two approximations of the Pareto front of a multi-objective optimization problem. We define $C(A, B)$ as the percentage of the solutions in *B* that are dominated by at least one solution in *A*, where $C(A, B)$ is denoted by C and $C(B, A)$ is denoted by C'.
- Let $\mathbf{S}$ be the set of final non-dominated points, and $r = (r_1, r_2, ..., r_m)^{\mathrm{T}}$ be a reference point in the objective space that is dominated by all Pareto optimal objective vectors. Then I_H is the volume of the region dominated by $\mathbf{S}$ and bounded by r[2].

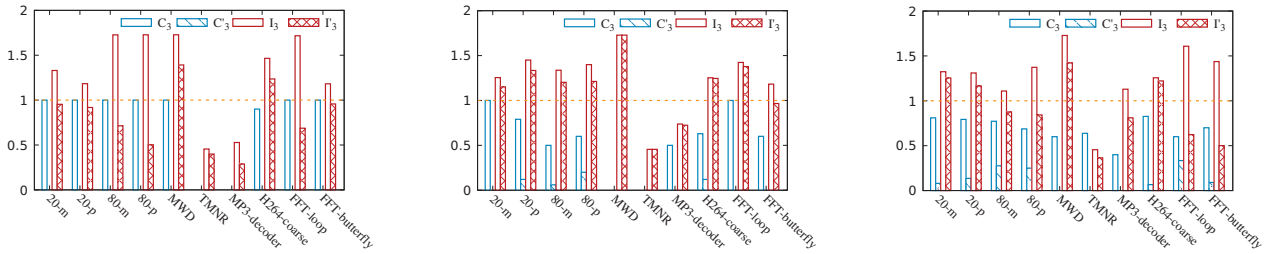
We consider the two metrics together. Informally speaking, the larger value of the same metric indicates better performance. We present three groups of comparison among the two metrics in Fig.4: C_1: with local search (I_1) and without it (I'_1); C_2: with task clustering (I_2) and with random initialization (I'_2), and C_3: with MOHA (I_3) and with NSGAII (I'_3). In the figure, zero valuation leads to a missing column.

Compared with the results of C_1 and C'_1 in Fig.4a, MOHA can find better solutions with the LS process. For TMNR, using LS or not can both find five solutions. Among the two groups of solutions, three of them are the same, and the other two have no dominance relation. Therefore, both C_1 and C'_1 are zero. However, the volume of using LS (I_1) is larger than that of without LS (I'_1), showing that the results of using LS are closer to the Pareto fronts. For MP3-decoder, the number of solutions found with LS is larger than the case without LS. However, there is no dominance relation either. Though the step of LS increases the duration of a genetic process, the quality of solutions with LS is better, as illustrated by the comparison between I_1 and I'_1.

The results in Fig.4b show that though random initialization can generate some non-dominated solutions ($C'_2 > 0$), and the difference of volumes is not quite obvious, many solutions without task clustering are dominated by those with the heuristic. For MWD and TMNR, both can find the same solutions and their C_2 and C'_2 (I_2 and I'_2) are the same. For MWD, TMNR and MP3-decoder, their scales are smaller, and the dependence relations are simple. Therefore, the advantage of applying the heuristics separately is not quite obvious in Fig.4a and Fig.4b.

In the third comparison in Fig.4c, the percentage of dominance with MOHA is much better than that with NSGAII. That is, by applying the heuristics, MOHA can always find more and better solutions. In randomly generated cases, the computation complexity of MOHA becomes higher with the increasing number of tasks, which spends more time in finding non-dominated solutions. Although NSGAII sometimes can find some solutions that dominate part of solutions with MOHA (where $C_3 \neq C'_3$ and both C_3 and $C'_3 > 0$), the vol-

[2]All the values are normalized into [0,1] for a fair comparison

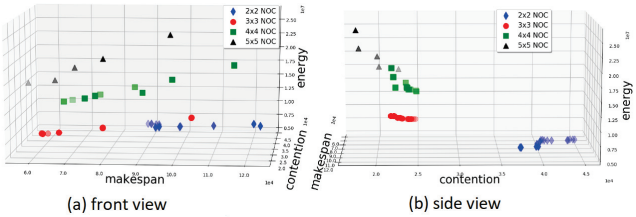


(a) local search (b) initialization (c) MOHA and NSGAII

Figure 4: Comparison between various heuristics and NSGAII

Table 4: Contention-aware strategy comparison

Case	$\|T\|$	$\|E\|$	SPC	CC	PaCC	PrCC
MWD	12	12	2666	2488	1878	1488
TMNR	14	16	2590	2340	1812	1800
H264-coarse	14	36	89743	86330	85329	83490
Mp3-decoder	16	16	33329	33253	33253	33253
FFT-loop	28	27	19219	12684	12351	12187
FFT-butterfly	32	48	17028	10523	10273	10189



(a) front view (b) side view

Figure 5: Platform specific solutions for fine-grained H264

ume with NSGAII is always smaller than that with MOHA. The fact demonstrates that the quality of solutions generated from MOHA is better than that from NSGAII.

Effectiveness for contention-aware optimization

To evaluate the effectiveness of Constraint 14 and Objective 22 in avoiding contention, we conduct the following comparison for real benchmarks: 1) with sequentializing all potential contention (SPC) in scheduling, by forbidding any overlapped communication whose routes involve shared links; 2) only with Constraint 14 (CC) that just separates nearly simultaneous task-to-task communication with shared links; 3) with path-contention minimization proposed in (Chou and Marculescu 2008) and CC (PaCC); and 4) with Objective 22 and CC (PrCC). That is, the comparison of the last three follows same mapping and scheduling constraints but various optimization goals. We employ makespan to evaluate their performance for real benchmarks.

In Table 4, the makespan of SPC is the largest. The makespan of PrCC is smaller than that of CC and PaCC in most cases. That is, without spatial contention reduction, only introducing latency to avoid contention (such as SPC and CC) is usually less efficient. In path-based contention minimization (PaCC), every communication path overlap is regarded as a contention, which may cause higher communication cost for scattered tasks, or lower parallel execution of tasks for communication cost reduction. Thus, its performance is worse than PrCC. For Mp3-decoder with few parallelizable tasks, the last three methods obtain the same result.

Evaluation on various platforms

We consider applying fine-grained H264 (with 264 tasks) to various NoC platforms (2×2, 3×3, 4×4 and 5×5, respectively). According to the distributions of solutions shown in Fig.5, we have the following observations. First, the increasing number of PEs leads to the increasing energy consumption, for the introduced communication and additional latency. Second, parallel execution of tasks on more available PEs shortens the makespan. Third, the smaller number of PEs results in higher degree of contention. However, for the 3×3 NoC platform, though from Fig.5(a) it is not easy to compare the valuations of makespan in the solutions with the cases of 4×4 and 5×5 platforms, the energy costs are smaller than those in the two cases. And as demonstrated in Fig.5(b), the values of contention in the solutions are smaller than the case of 2×2 platform. Therefore, the 3×3 platform may be a trade-off between parallel computation and the introduced communication cost.

Conclusion

We address the mapping and scheduling problem on NoC-based platforms for both time and energy efficient contention-aware applications. In mapping, contention degree (the proportion of shared physical links to the product of actual lengths) is evaluated and minimized to reduce spatial contention. In scheduling, a non-overlapping criterion is applied to separate potential contentions. For makespan and energy minimization, we propose a clustering heuristic to organize related tasks for mapping, and a capacity sensitive heuristic to balance the allocated tasks to a PE. To accelerate convergence, we further integrate a local search into the genetic process. The extensive experimentation validates the effectiveness of the constraint model, and the efficiency of MOHA in optimizing real applications.

Acknowledgments

This work has been partly funded by Key Research Program of Frontier Sciences, CAS, under Grant No.

QYZDJ-SSW-JSC036, the CAS-INRIA major project under No.GJHZ1844, NSFC under Grant No.61976050, and project of Jilin Provincial Science and Technology Department under Grant No. 20190302109GX.

References

Bolanos, F.; Rivera, F.; Aedo, J. E.; and Bagherzadeh, N. 2013. From UML specifications to mapping and scheduling of tasks into a NoC, with reliability considerations. *JSA* 59(7):429–440.

Chai, S.; Li, Y.; Wang, J.; and Wu, C. 2014. A list simulated annealing algorithm for task scheduling on network-on-chip. *JCP* 9(1):176–182.

Chou, C.-L., and Marculescu, R. 2008. Contention-aware application mapping for network-on-chip communication architectures. In *ICCD*, 164–169.

CPLEX, IBM ILOG. 2009. V12. 1: User manual for CPLEX. *International Business Machines Corporation* 46(53):157.

De Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *TACAS*, 337–340. Springer.

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6(2):182–197.

Dick, R. P.; Rhodes, D. L.; and Wolf, W. 1998. TGFF: task graphs for free. In *CODES/ CASHE*, 97–101. IEEE Computer Society.

He, O.; Dong, S.; Jang, W.; Bian, J.; and Pan, D. Z. 2012. Unism: Unified scheduling and mapping for general networks on chip. *TVLSI* 20(8):1496–1509.

Hu, J., and Marculescu, R. 2003. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *ASP-DAC*, 233–239.

Li, D., and Wu, J. 2016. Energy-efficient contention-aware application mapping and scheduling on NoC-based MPSoCs. *JPDC* 96:1–11.

Mehran, A.; Saeidi, S.; Khademzadeh, A.; and Afzali-Kusha, A. 2007. Spiral: A heuristic mapping algorithm for network on chip. *IEICE Electronics Express* 4(15):478–484.

Nedjah, N., and de Macedo Mourelle, L. 2014. Application mapping in network-on-chip using evolutionary multi-objective optimization. In *Hardware for Soft Computing and Soft Computing for Hardware*. Springer. 155–171.

Neubauer, K.; Wanko, P.; Schaub, T.; and Haubelt, C. 2018. Exact multi-objective design space exploration using ASPmT. In *DATE*, 257–260. IEEE.

Sahu, P. K., and Chattopadhyay, S. 2013. A survey on application mapping strategies for network-on-chip design. *JSA* 59:60–76.

Tino, A., and Khan, G. N. 2011. Multi-objective tabu search based topology generation technique for application-specific network-on-chip architectures. In *DATE*, 1–6.

Yang, L.; Liu, W.; Jiang, W.; Li, M.; Yi, J.; and Sha, E. H.-M. 2016. Application mapping and scheduling for network-on-chip-based multiprocessor system-on-chip with fine-grain communication optimization. *TVLSI* 24(10):3027–3040.

Zitzler, E., and Thiele, L. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evol. Comput.* 3(4):257–271.

Zitzler, E.; Thiele, L.; Laumanns, M.; Fonseca, C. M.; and Da Fonseca, V. G. 2003. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Trans. Evol. Comput.* 7(2):117–132.